Frame Fields for Hexahedral Mesh Generation

Inauguraldissertation

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Heng Liu

aus China

Leiter der Arbeit:
Prof. Dr. David Bommes
Institut für Informatik

Frame Fields for Hexahedral Mesh Generation

Inauguraldissertation

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Heng Liu

aus China

Leiter der Arbeit:
Prof. Dr. David Bommes
Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 17.04.2023

Der Dekan
Prof. Dr. M. Herwegh

Acknowledgement

I would like to express my sincere gratitude to multiple people and institutions for their kind support during my thesis work.

First of all, my most profound appreciation goes to my advisor Prof. David Bommes for his invaluable guidance, support, and encouragement throughout my Ph.D. journey, without whom this work would have never been accomplished. His expertise and insights have been instrumental in shaping my research, and his way of thinking and pursuit of perfection in research have had a long-term impact on me.

I would like to thank my external evaluator, Prof. Jin Huang, for taking precious time to evaluate my thesis work and providing constructive feedback and valuable suggestions.

Additionally, I would like to acknowledge the European Research Council and the Chinese Scholarship Council that funded this research. Their generous support is essential for completing my research and thesis.

I am also grateful to work within the Computer Graphics Group at the University of Bern, AICES, and VCI at RWTH Aachen for the excellent working environment. I would like to thank all the current and former colleagues, in particular Martin Heistermann for the maintenance of hardware and technical supports, Pierre-Alexandre Beaufort for his valuable feedback on my thesis, Denis Kalmykov for the proofreading, and Nicolas Gallego-Ortiz, Simone Raimondi, Valentin Nigolian, and Liubov Kamaldinova for inspiring discussions in our free-fairs and other scientific activities. Thanks should also go to colleagues at MIT, Justin Solomon, Paul Zhang, and Edward Chien, for the collaboration.

Finally, many thanks to my family, especially my parents, whom I miss so much. Although not able to see each other throughout the pandemic, your love, support, and encouragement keep me motivated and strong. I would be remiss in not mentioning all my close friends, particularly Mo Wa Wong, for their accompany and caring during this incredible Ph.D. journey.

Abstract

As a discretized representation of the volumetric domain, hexahedral meshes have been a popular choice in computational engineering science and serve as one of the main mesh types in leading industrial software of relevance. The generation of high quality hexahedral meshes is extremely challenging because it is essentially an optimization problem involving multiple (conflicting) objectives, such as fidelity, element quality, and structural regularity. Various hexahedral meshing methods have been proposed in past decades, attempting to solve the problem from different perspectives. Unfortunately, algorithmic hexahedral meshing with guarantees of robustness and quality remains unsolved.

The frame field based hexahedral meshing method is the most promising approach that is capable of automatically generating hexahedral meshes of high quality, but unfortunately, it suffers from several robustness issues. Field based hexahedral meshing follows the idea of integer-grid maps, which pull back the Cartesian hexahedral grid formed by integer isoplanes from a parametric domain to a surface-conforming hexahedral mesh of the input object. Since directly optimizing for a high quality integer-grid map is mathematically challenging, the construction is usually split into two steps: (1) generation of a feature-aligned frame field and (2) generation of an integer-grid map that best aligns with the frame field. The main robustness issue stems from the fact that smooth frame fields frequently exhibit singularity graphs that are inappropriate for hexahedral meshing and induce heavily degenerate integer-grid maps. The thesis aims at analyzing the gap between the topologies of frame fields and hexahedral meshes and developing algorithms to realize a more robust field based hexahedral mesh generation.

The first contribution of this work is an enumeration of all local configurations that exist in hexahedral meshes with bounded edge valence and a generalization of the Hopf-Poincaré formula to octahedral (orthonormal frame) fields, leading to necessary local and global conditions for the hex-meshability of an octahedral field in terms of its singularity graph. The second contribution is a novel algorithm to generate octahedral fields with prescribed hex-meshable singularity graphs, which requires the solution of a large non-linear mixed-integer algebraic system. This algorithm is an important step toward robust automatic hexahedral meshing since it enables the generation of a hex-meshable octahedral field.

In the collaboration work with colleagues [BRK+22], the dataset HexMe consisting of practically relevant models with feature tags is set up, allowing a fair evaluation for practical hexahedral mesh generation algorithms. The extendable and mutable dataset remains valuable as hexahedral meshing algorithms develop. The results of the standard field based hexahedral meshing algorithms on the HexMesh dataset expose the fragility of the automatic pipeline.

The major contribution of this thesis improves the robustness of the automatic field based hexahedral meshing by guaranteeing local meshability of general feature aligned smooth frame fields. We derive conditions on the meshability of frame fields when feature constraints are considered, and describe an algorithm to automatically turn a given non-meshable frame field into a sim-

ilar but locally meshable one. Despite the fact that local meshability is only a necessary but not sufficient condition for the stronger requirement of meshability, our algorithm increases the 2% success rate of generating valid integer-grid maps with state-of-the-art methods to 57%, when compared on the challenging HexMe dataset.

Contents

Chapter 1

Introduction

In computational engineering science (CES), engineers and scientists build computational models and run simulations on computers to solve the relevant questions that emerge from intricate physical problems in engineering analysis and design, as well as natural phenomena in scientific research. With this technology, engineers and scientists can work in areas where either traditional experiments are unrealistic to conduct, or an unbearable amount of resources need to be invested. Hence, CES has a wide range of applications, to name just a few, aerospace engineering, biology and medicine, civil engineering, and physics. For example, structural design and material compositions can be examined in computational engineering before manufacturing by simulating the car crash test. It significantly shortens the development process, which usually takes several iterations until a satisfactory solution is obtained.

In many CES techniques, a key component is to faithfully discretize 3D geometric objects into volumetric meshes which carry physical properties, such as mechanical properties of materials. A volumetric mesh is a network of interconnected polyhedral elements that fit together seamlessly to represent the surface and interior of the object. Volumetric meshes can be classified on the type of polyhedral elements, e.g., hexahedral mesh composed of hexahedral elements. Element type often strongly impacts the accuracy and efficiency of the computation. Hexahedral (hex) and tetrahedral (tet) elements are the most prominent elements used for discretizing volumes, as depicted in Fig.1.1. Which element type is better (tet vs. hex) has been a longstanding controversy, and the debate is still ongoing [SHG+22]. Researchers have done many experiments to compare the performance of these two element types in different scenarios. While the conclusions are problem dependent, it is commonly believed that the linear hexahedral element outperforms the linear tetrahedral elements, making it preferable in applications that use linear elements [GP21]. Particularly when working with basis functions of high polynomial degree, e.g. in Spectral Element Methods [Kop09], or when requiring higher-order continuity between cells like in Isogemetric Analysis [CRBH06], the tensor-product nature of hexahedral meshes offers advantages, for instance, a better performance-to-accuracy tradeoff. Both academia and industry show high interest in hexahedral meshes. The number of scientific articles on hexahedral meshes has been increasing over the years [BRK+22]. Users of the mainstream CAD/CAE software, e.g., [Alt22, ANS22, Cor22, CUB22], demand high quality hexahedral

meshes for their computational tasks.

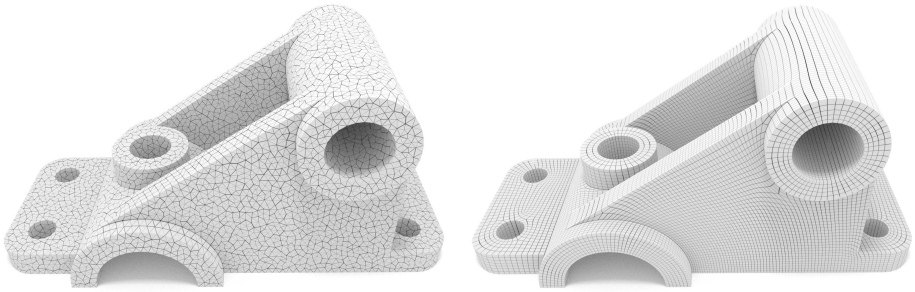


Figure 1.1: A tetrahedral mesh (left) and a hexahedral mesh (right) of an industry model.

Unlike tetrahedral meshing, for which multiple automatic and robust algorithms are available [HSW+20, ST06, FP09], so far there is no automatic algorithm capable of robustly generating high quality hexahedral meshes for general geometric shapes [PCS+22]. The challenge of hexahedral meshing lies in the robustness and automation of the generation algorithm, as well as multiple objectives determining a "high quality" hexahedral mesh which are sometimes conflicting, as we will see in Chapter 2. Although various algorithms have been proposed in the last decades, automatically and robustly generating high quality hexahedral meshes remains an open problem. Most techniques used in industry can produce high quality hexahedral meshes, but with substantial manual effort. Some algorithms, like grid-based methods, can robustly generate hexahedral meshes, while the mesh quality is usually not satisfactory, especially near the boundary of the object. A more thorough comparison of different algorithms will be discussed in Chapter 3. Among all categories of hexahedral meshing algorithms, frame field based hexahedral meshing shows the potential to automatically produce all-hex meshes of high quality. Therefore, this thesis mainly focuses on automatic hexahedral meshing algorithms based on frame fields.

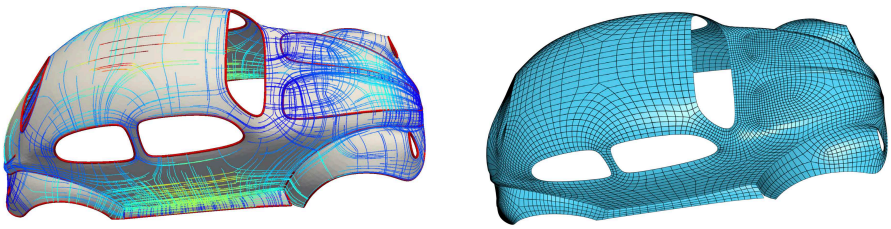
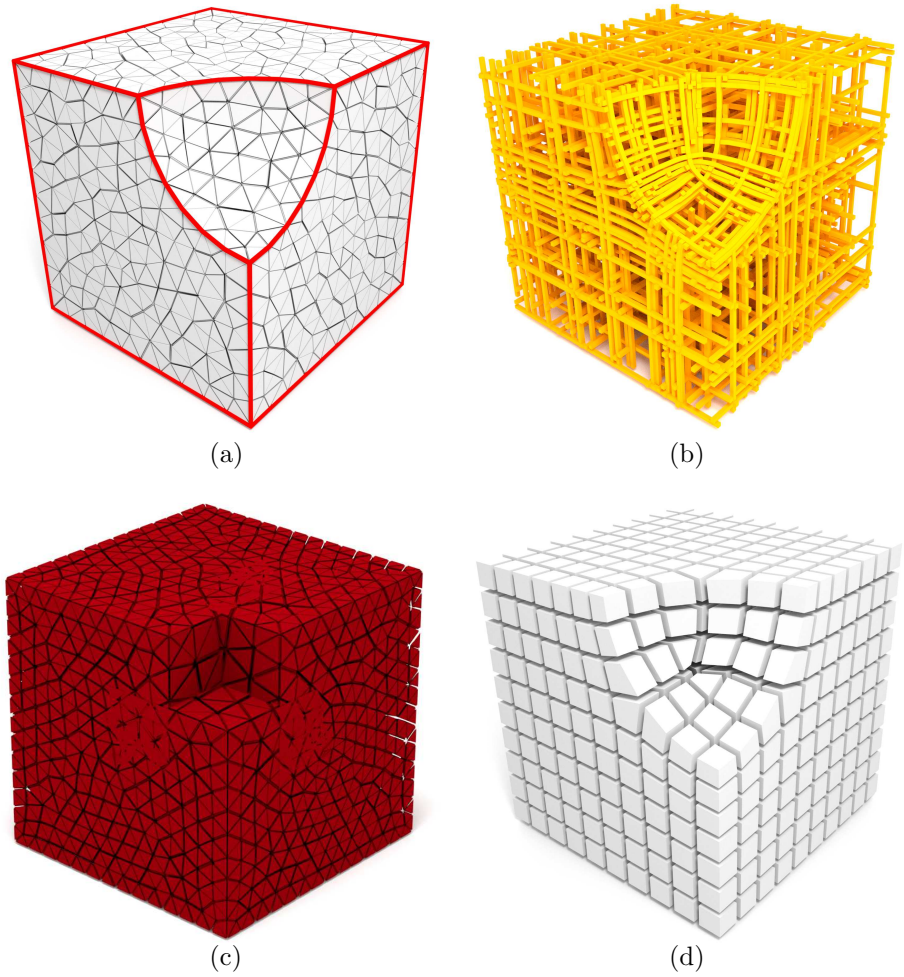


Figure 1.2: 2D frame field (left) and the resulting quadrilateral mesh (right) of the *BEETLE* model. [JFH+15]

As a 2D counterpart, algorithms of quadrilateral (quad) meshing for surface objects with 2D frame fields have demonstrated great success recently (refer to Fig.1.2). Frame field based hexahedral meshing naturally extends the idea to 3D volumes, sharing the common core concept: taking a 2D/3D frame field as guidance for computing an integer-grid map f from the geometric domain to

the parametric domain: $f : \mathbb{R}^3 \to \Omega$. The standard pipeline of frame field based hexahedral meshing consists of the following steps, see Fig.1.3 for an illustration:

1. Specification of frame field alignment constraints.
2. Feature aligned smooth frame-field generation. [RSL16]
3. Integer-grid map (IGM) generation guided by the frame field. [NRP11]
4. Hexahedral mesh extraction from the IGM paramterization. [LBK16]



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

Figure 1.3: Standard frame field based hexahedral meshing pipeline: (a) alignment constraints: frame field tangent to the feature arcs (red) and the surface normals of the tetrahedral mesh, (b) frame field visualized as streamlines, (c) IGM paramterization, (d) the final hexahedral mesh extracted from the parameterization.

Often frame field based hexahedral meshing delivers hexahedral elements of low geometric distortion in the majority of the volume owing to the smoothness of frame fields. Moreover, smoothly deformed frame fields under alignment constraints yield naturally arising singularity graphs that enable hexahedral meshes

with good block structure. Importantly, frame field based hexahedral meshing supports alignment constraints for boundaries as well as interior features of the geometric domain, enabling feature-aligned hexahedral meshes. In many applications, simulations significantly benefit from meshes aligned to directions [MUF21].

Despite many advantages, frame field based hexahedral meshing suffers from robustness issues, which are rooted in two aspects: (i) a 3D frame field with a topology that is not hex-meshable, or meshable for short, and (ii) the inability to guarantee local injectivity for volumetric maps. The 2D frame fields, where singularities are isolated points around which the field can circulate in the domain, are already well-understood. The seminal work of Myles et al. [MPZ14] presents an algorithm that is capable of converting an input 2D frame field on a triangular mesh into a (globally) meshable one. Additional theoretical guarantees have been identified in [VO19]. Much less is known about the 3D frame fields, where singularities form a graph consisting of singular arcs and nodes that determine their topological structure. The set of frame field singular graphs is a significantly larger superset of singular graphs of hexahedral meshes. The gap in between is the central issue that causes failure in downstream steps of the hexahedral meshing pipeline, especially collapses in the integer-grid map, consequently the most crucial challenge is to understand the discrepancy and to transform smooth frame fields into meshable frame fields. In addition, given a topologically meshable frame field, there is no guarantee that a valid seamless paramterization can be found due to (i) the non-convexity of the optimization problem and (ii) lacking degrees of freedom during the piecewise linear deformation. Although various techniques [GKK$^+$21, RPPSH17] have been proposed in the past years, seeking degenerate-free maps remains unsolved for challenging cases.

With these issues in mind, the main topic of this thesis is to improve the robustness of the frame field based hexahedral meshing pipeline. It first studies the fundamental questions on the compatibility between the hexahedral mesh topology and the frame field topology in terms of singularity graphs. A detailed analysis of local hexahedral mesh topology is presented in Chapter 2, which implies local conditions for frame fields to be locally meshable. However, finding sufficient conditions of the global meshability of frame fields is theoretically much more challenging, and thus transforming an arbitrary frame field into a meshable frame field is beyond the scope of this work. Alternatively, provided a user-designed singularity graph, an efficient algorithm to reconstruct the corresponding frame field is described in Chapter 5. Although it requires the user's expertise on globally meshable singularity graphs, it also provides flexibility to users to control the hexahedral mesh structure. Secondly, targeting for a consistent evaluation of automatic hexahedral meshing algorithms, the tetrahedral mesh dataset - HexMe, with feature tags, is presented in Chapter 7. Note that this is a collaboration work with colleagues. The evaluation results of the standard field based hex meshing pipeline reveals severe robustness issues. Most importantly, aiming for a fully automatic hexahedral meshing pipeline, a suitable approach is to start from automatically generated feature-aligned smooth frame fields and repair the incompatibilities towards meshable frame fields. This thesis further explores the direction of non-meshable 3D frame field topology with feature constraints considered. Previous correction algorithms focus solely on the meshability of singular arcs [LLX$^+$12, JHW$^+$14] but ignore issues that

occur at singular nodes, and certain feature configurations. In Chapter 7, we explain why a certain class of non-meshable singular nodes, called *zipper nodes*, frequently exist in smoothness-optimized frame fields. This accounts for the low success rate of state-of-the-art algorithms, specifically for models with non-trivial feature constraints. We propose repair operations for local defects in piecewise constant 3D frame fields and design a practical algorithm to realize the local meshability of frame fields. With locally meshable frame fields, the field based hexahedral meshing pipeline achieves a significantly higher success rate on HexMe the previous state-of-the-art methods.

Specifically, the contributions of this thesis are as follows:

- *Singularity-Constrained Octahedral Fields for Hexahedral Meshing.* [LZC+18] This work enumerates all possible local configurations w.r.t. the topology of hexahedral meshes, given that the edge valence is restricted to a specific range. Based on this observation, we further generalize the Hopf-Poincaré formula to octahedral fields, showing necessary local and global conditions for a hexahedral meshable octahedral field w.r.t. its singularity graph. Secondly, this thesis proposes an efficient algorithm to produce smooth octahedral fields that match the prescribed hexahedral meshable singularity graphs, which can then be fed to subsequent steps of the hexahedral meshing pipeline. While manual effort is required for the input singularity graph, it allows users to control the topology of the output hexahedral meshes.

- *Hex Me If You Can* [BRK+22] This collaboration with colleagues intends to collect meshes with varying complexity, enabling consistent and practically meaningful evaluation of hexahedral meshing algorithms and related techniques, specifically regarding the correct meshing of specified feature points, curves, and surfaces. HexMe is an extendable dataset comprising 189 tetrahedral meshes with tagged features and a workflow to generate them. 63 computer-aided design (CAD) models from various databases, classified into three categories (simple, nasty, and industrial), are tetrahedralized with Gmsh into three kinds: uniform, curvature-adapted, and box-embedded. The mesh generation pipeline is defined with the help of Snakemake, a modern workflow management system that allows us to specify a fully automated, extensible, and sustainable workflow. The HexMe dataset is built with evolution in mind and prepared for future developments.

- *Locally Meshable Frame Fields*

  This work aims at locally meshable frame fields which align with feature edges and normals of feature faces. We show possible sectors in 2D frame fields and that only quad-sectors exists in quadrilateral mesh induced 2D frame fields. For 2D frame fields, we discuss strategies to repair other invalid sectors to meshable ones locally and to ensure global meshability. For 3D frame fields, we first analyze local meshable conditions of singular arcs and nodes where multiple singular arcs interact. Once the meshability of singular arcs is ensured with techniques similar to those of 2D singularities, singular nodes can be algorithmically decomposed into fundamental components with the *arc zipping* operation. We propose a

series of local repair operations for piecewise constant frame fields embedded in tet meshes. We also design a practical algorithm to convert feature-aligned smooth frame fields to similar but locally meshable ones. Despite, the algorithm can only guarantee local meshability, which is necessary but not sufficient for meshability, the evaluation on the basis of the challenging HexMe dataset [BRK+22] shows that our algorithm is able to construct valid integer-grid maps for 57% of the domains and therefore is significantly more robust than state-of-the-art frame field based methods, which only succeed for 2% of the inputs. An additional contribution is a novel algorithm to optimize seamless maps and integer-grid maps for a given frame field, which is more robust than the available state-of-the-art techniques. The content regarding this part is summarized in our "Locally Meshable Frame Fields" paper and has been conditionally accepted as a Journal Paper to SIGGRAPH 2023.

This thesis is in the context of the ERC (European Research Council) Starting Grant project AlgoHex (Grant agreement No. 853343), short for Algorithmic Hexahedral Mesh Generation. The ultimate goal of the AlgoHex project is to develop hexahedral meshing algorithms for general geometric shapes which are robust, scalable, and controllable w.r.t. mesh quality requirements. This work makes a theoretical breakthrough on the fundamental scientific challenges regarding frame field topology alongside an open source c++ library *libAlgoHex* (`http://www.algohex.eu`), which implements our advanced frame field based hexahedral meshing pipeline.

Chapter 2

Hexahedral Meshes

A hexahedral mesh is a representation of a geometric domain with conforming topological cubes. Both topology and geometry of hex meshes are highly relevant in either hex mesh generation and processing algorithms or various practical applications in CES. We begin by introducing basic concepts about hex mesh topology from the singularity point of view since it uniquely characterizes the topology of a hex mesh. As one of this thesis's contributions, we prove that only a finite number of one-ring structures exist in hex meshes assuming restricted edge valence, and enumerate all local configurations at hex mesh vertices. Understanding local hex mesh topology helps identify the fundamental problems in frame field based hex meshing and tackle the underlying difficulties. From a higher-level perspective than the local topology, we next discuss the regularity of the hex mesh structure, followed by a short description of the geometric quality of hex meshes.

2.1 Hexahedral Mesh Topology

A hex mesh $\mathcal{H} = (V, E, Q, H)$ is a CW-complex [HPM02], which decomposes a volumetric region $M \subset \mathbb{R}^3$ into hexahedral cells H, formed by quadrilaterals Q, edges E, and vertices V. The *boundary* $\partial\mathcal{H} \subset (V, E, Q)$ consists of all quadrilaterals ∂Q incident to only one hexahedron, and includes all vertices ∂V and edges ∂E of ∂Q. All remaining mesh elements are referred to as *interior* elements. A hex mesh example is shown in Fig. 2.1.

2.1.1 Local Topology of Edges

An interior edge is called *regular* if it is incident to exactly four hexahedra; otherwise it is *singular*. Similarly, a boundary edge is regular if incident to exactly two hexahedra, and otherwise it is singular. The *hexahedral valence* $\mathrm{val}_h(\sigma) \in \mathbb{N}_{\geq 1}$ of a mesh element $\sigma \in V \cup E \cup Q$ is the number of its incident hexahedra. Based on this, the *index* $\mathrm{idx}(e)$ precisely specifies the topological type of an edge e by measuring its deviation from regularity:

$$\mathrm{idx}(e) = \begin{cases} (4 - \mathrm{val}_h(e))/4 & \text{for interior } e \\ (2 - \mathrm{val}_h(e))/4 & \text{for boundary } e. \end{cases} \tag{2.1}$$

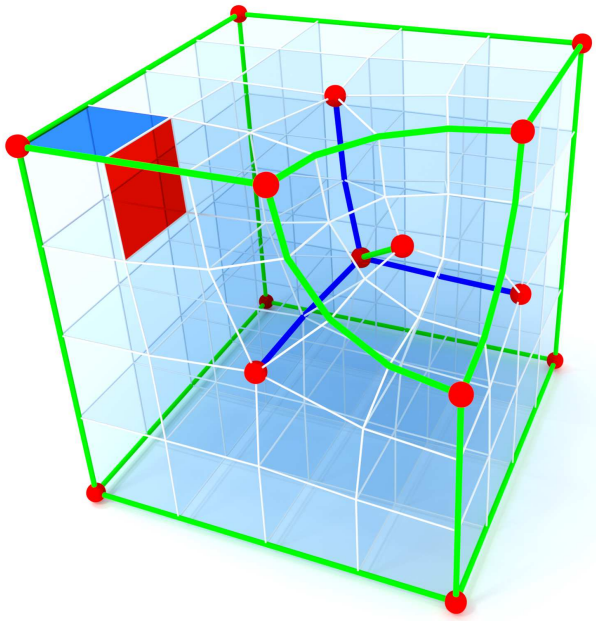


Figure 2.1: Hex mesh example. Green and blue edges indicate valence 3 and 5 singularities (valence 1 on boundary). Red vertices indicate singular vertices (interior and boundary). The dark blue and red face indicate two faces of a hexahedron, blue on the boundary, red inside.

For interior edges, this is equivalent to the fractional index of the extruded quad mesh singularity, present in the Hopf-Poincaré formula for 2D frame fields [RVLL08].

In principle, there are infinitely many topological configurations since the valence $\mathrm{val}_h(e) \in \mathbb{N}_{\geq 1}$. Considering the quality of the geometric embedding, however, only a very small subset is practically relevant for hex meshing, as depicted in Fig. 2.2. For interior edges, only those with hexahedral valence 3, 4 and 5 corresponding to indices from $I_{\text{interior}} = \{1/4, 0, -1/4\}$ are typically desirable since other cases induce lower-quality scaled Jacobians. Moreover, all such low-quality cases could be easily split by sheet insertion [MESB08] into multiple edges of the three good quality cases (cf. Fig. 2.3). The same argument holds for boundary edges, where the practically important set is given by hexahedral valences of 1, 2, 3 and 4 corresponding to indices $I_{\text{boundary}} = \{1/4, 0, -1/4, -1/2\}$ (cf. Fig. 2.2). A simply-connected mesh with all interior edges being regular is the pullback of the Cartesian grid under a continuous locally-injective (on the interior) map of the volume into $\mathbb{R}^3$ (cf. Fig. 4.2). If this map is also globally injective, then the mesh is homeomorphic to a subset of the Cartesian grid and is often referred to as a *polycube mesh* [GSZ11].

2.1.2 Singularity Graph

The subset of singular edges $E_S \subset E$ forms a graph, which can be conformingly partitioned into segments of singular edges with identical index, referred to as *singular arcs*. Singular arcs are either closed or bounded by *singular nodes*,

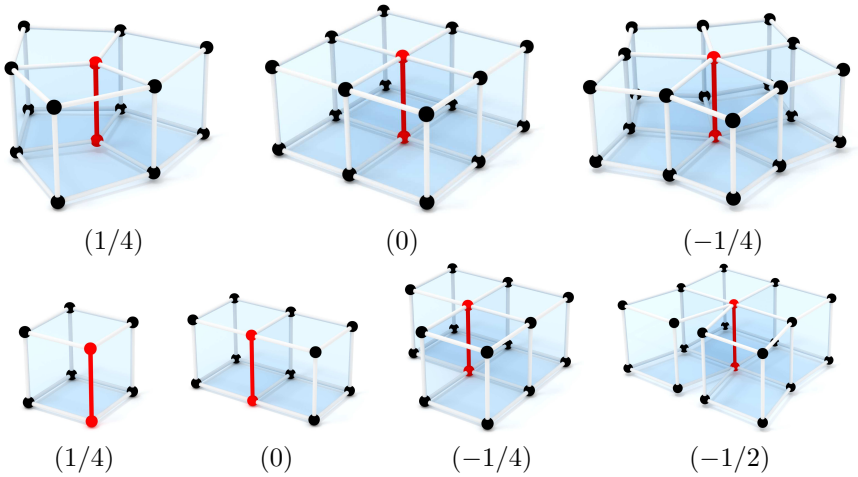


Figure 2.2: Practically relevant hex mesh edges, highlighted in red. The top row illustrates interior edges of valence 3, 4, and 5. The bottom row illustrates boundary edges of valence 1, 2, 3, and 4.

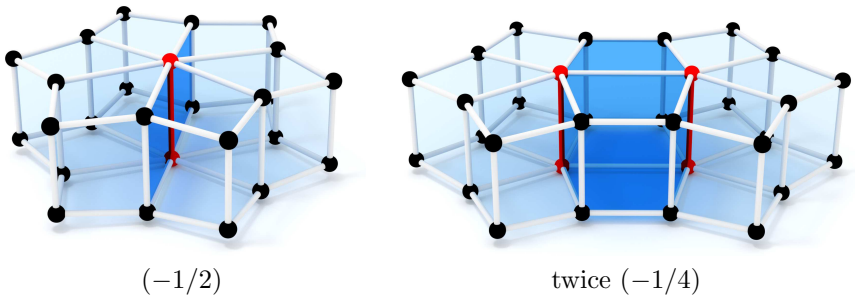


Figure 2.3: (Left) Interior singular edge of valence 6 in red. Dark blue faces indicate the location of dual sheet insertion. (Right) Hex mesh after dual sheet insertion. The single valence 6 edge is split into two valence 5 edges.

where they split into multiple singular arcs or terminate at the boundary. Together, the singular arcs and singular nodes $V_S \subset V$ form the *singularity graph* $\mathcal{S} = (V_S, E_S)$ of the hex mesh (cf. Fig. 2.1). Tracing sheets from all edges of the singularity graph results in the *base complex* of the hex mesh, which is the coarsest partitioning of the mesh into regular blocks [GDC15].

A proposed embedded singularity graph $\mathcal{S} = (V_S, E_S)$ for an input region $M \subset \mathbb{R}^3$ is *globally meshable* if there is a hex mesh of M with singularity graph matching $\mathcal{S}$. Similarly, we say that $\mathcal{S}$ is *locally meshable* if there is a hex mesh of the neighborhood of $\mathcal{S}$, with singularity graph containing $\mathcal{S}$ as a subgraph. This is equivalent to having local type assignments for elements of V_S, E_S. For arcs, this is an index, as described above, and for nodes, valid types are described below. A more general definition on local meshability of frame fields will be discussed in Chapter 7.

2.1.3 Local Topology of Vertices

A critical theoretical consideration for design of hex meshing algorithms is determining which topological types of vertices exist in a hex mesh. Compared

to the simple quarter-integer index for edges the answer is more complicated. Following Nieser et al. [NRP11], interior vertices of hex meshes topologically correspond to triangulations of the sphere. The idea is to intersect the neighborhood of a hex mesh vertex with a sphere, providing the correspondence of hex mesh edges, faces, and cells with vertices, arcs, and triangular patches on the sphere (see Fig. 2.4). Consequently, from a topological point of view, hex meshes admit infinitely many different vertex topologies. However, restricting the incident edges to the practically relevant set of hexahedral valences 3, 4 and 5 results in only 11 topologically different interior vertex types. These are illustrated in Fig. 2.5, and a detailed topological analysis is provided below.

Proof. A triangulation of the sphere with vertices V, edges E, and faces F necessarily obeys Euler's formula: $|V| - |E| + |F| = 2$. As all faces are triangles, we have $3\,|F| = 2\,|E|$; and also that $\sum_{v \in V} \deg v = 2\,|E|$, resulting in the following form:

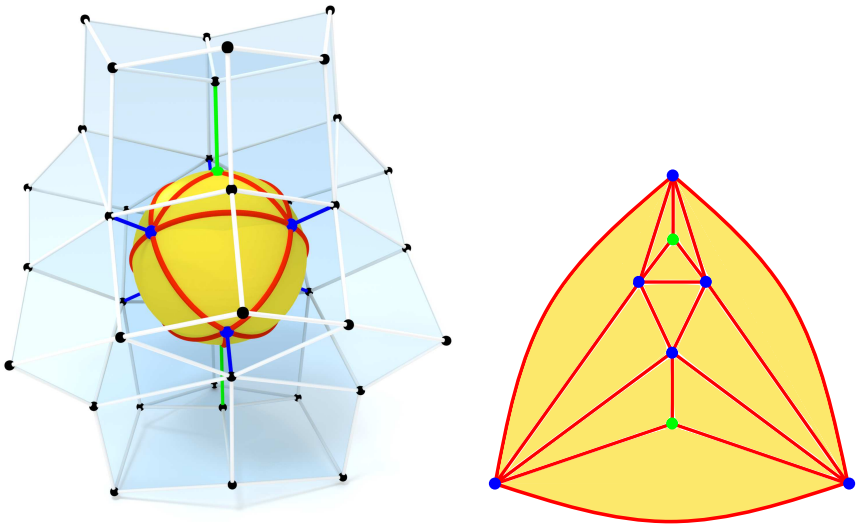
$$\frac{1}{6} \sum_{v \in V} (6 - \deg v) = 2 \;\Leftrightarrow\; 3i + 2j + k = 12, \tag{2.2}$$

where i, j, k denote the number of valence $3, 4, 5$ vertices in the triangulation. Let the *signature* of a triangulation denote the triplet (i, j, k). As i, j, k are all nonzero integers, there are a finite number of solutions, or possible signatures for such a triangulation, listed below, organized by the total number of vertices $|V| = i + j + k$:

$\|V\| = 4$:	(4,0,0)			
$\|V\| = 5$:	~~(3,1,1)~~	(2,3,0)		
$\|V\| = 6$:	~~(3,0,3)~~	(2,2,2)	~~(1,4,1)~~	(0,6,0)
$\|V\| = 7$:	(0,5,2)	(1,3,3)	~~(2,1,4)~~	
$\|V\| = 8$:	(0,4,4)	~~(1,2,5)~~	(2,0,6)	
$\|V\| = 9$:	(0,3,6)	~~(1,1,7)~~		
$\|V\| = 10$:	(0,2,8)	~~(1,0,9)~~		
$\|V\| = 11$:	~~(0,1,10)~~			
$\|V\| = 12$:	(0,0,12)			

The signatures that have been struck out, correspond to signatures with no valid corresponding triangulation of the sphere. This is a consequence of previous work [SH77, MSCR07] which characterizes the possible vertex degrees of planar (and hence sphere) triangulations. The remaining signatures are unique corresponding local singularities, see Fig. 2.5. □

Similarly, boundary vertices can be classified by intersecting the neighborhood of a hex mesh vertex with a hemisphere, providing the topological equivalence of boundary hexahedral vertices with triangulations of the disc (see Fig. 2.6). Again, there are infinitely many such triangulations, but we restrict to a practically relevant subset of those with incoming boundary edges of hex valence 1, 2, 3, and 4; incoming interior edges of hex valence 3, 4, and 5; and fewer than 9 incident hexahedra. This results in 237 topologically different singular boundary vertices, which can be found using an exhaustive enumeration algorithm that is describe in [LZC+18]. We use $\mathcal{H}_V$ to denote this set of practically relevant interior and boundary vertex singularity types. A locally meshable valence-$\{3, 4, 5\}$ singularity graph must have all nodes be within this



(a) (b)

Figure 2.4: (a) Interior singular vertex intersected with a yellow sphere. The intersection of the sphere with hex mesh faces is shown in red. (b) Planar representation of the triangulation of the sphere depicted on the left. Vertices correspond to intersections of the sphere with hex mesh edges.

set. This chosen subset of hexahedral vertex topologies coincides with the requirements of many applications but might be inappropriate for others. Unless otherwise noted, our discussions and algorithms are in general not restricted to this choice and could be easily extended to other finite subsets specified by bounds on edge valence and number of incident hexahedra. It is clear that for any application caring about the shape of hexahedra there is an upper bound on the number of hexahedra incident at a vertex, since packing an increasing number of hexahedra into the 4π solid angle of $\mathbb{R}^3$ necessarily deteriorates the worst scaled Jacobian.

2.1.4 A Global Necessary Condition

The above classifications define local hexability for a singularity graph. Additionally, we have a global necessary condition for global hexability which is simple to state and check:

$$\sum_{v \in \partial V_{\mathcal{S}}} \frac{1}{2}\left(1 - \frac{\mathrm{val}_h(v)}{4}\right) - \sum_{e \in \partial E^-_{\mathcal{S}}} \mathrm{idx}(e) + \sum_{v \in \mathring{V}_{\mathcal{S}}} \left(1 - \frac{\mathrm{val}_h(v)}{8}\right) - \sum_{e \in \mathring{E}^-_{\mathcal{S}}} \mathrm{idx}(e) = 0, \quad (2.3)$$

where $\partial V_{\mathcal{S}}, \mathring{V}_{\mathcal{S}}$ denote the boundary and interior nodes of the singularity graph, and $\partial E^-_{\mathcal{S}}, \mathring{E}^-_{\mathcal{S}}$ denote the non-closed boundary and non-closed interior singular arcs. This condition is the analogue of the discrete Hopf-Poincaré formula for

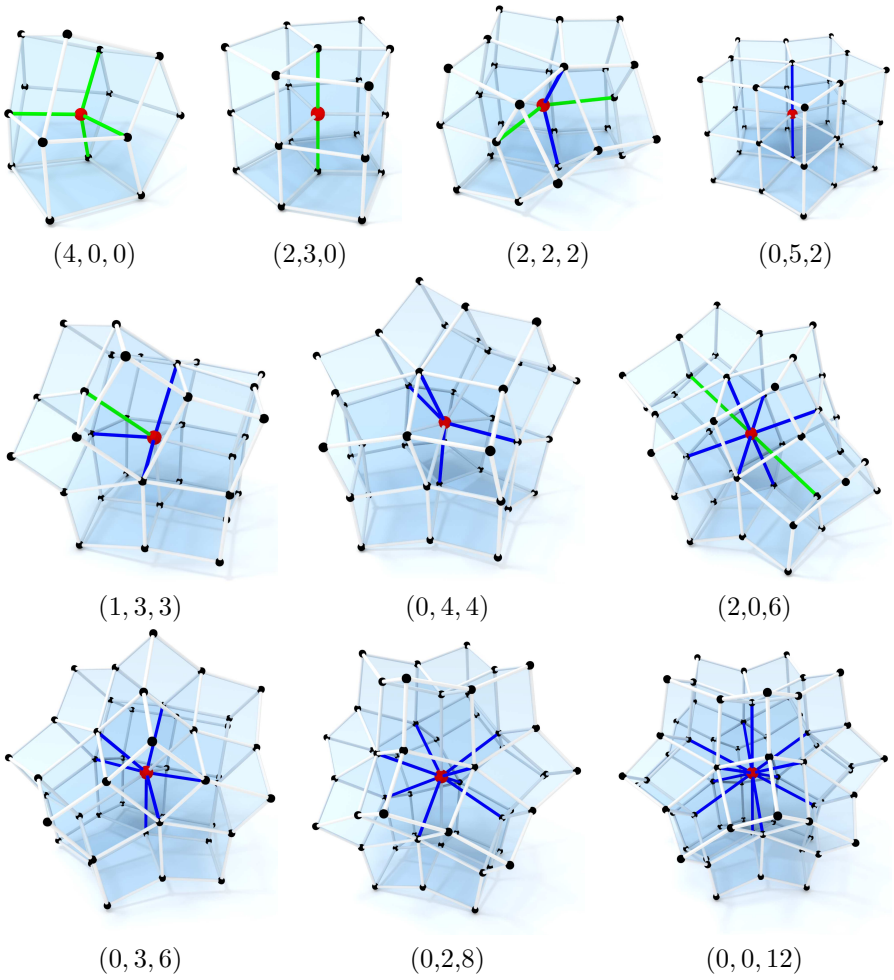
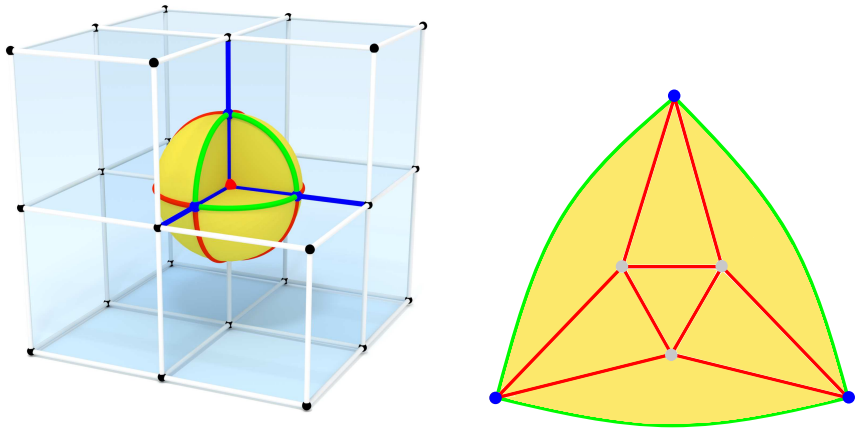


Figure 2.5: Singular Node Types. Signatures of the nodes are shown below. Green and blue edges are of valence 3 and 5 respectively.

quad meshes:

$$\sum_{v \in \partial V} \mathrm{idx}(v) + \sum_{v \in V \setminus \partial V} \mathrm{idx}(v) = \chi(S), \tag{2.4}$$

where $\mathrm{idx}(v)$ is defined as in Eq. (2.1) (with substitution of quad valence for hex valence), and S is the quad-meshed surface. Condition (2.3) can be derived with a simple combinatorial counting argument, given in the appendix of [LZC+18]. This condition holds for global hex meshes with arbitrary edge valence, as well as for boundary-aligned locally meshable frame fields—even those not globally meshable. The second generalization is analogous to the fact that Eq. (2.4) holds for 2D frame fields on surfaces, even if they are not globally quad-meshable. These hex-meshability definitions for fields are discussed in further detail in §4.3 and a proof for the generalization to locally meshable frame fields can be found in the supplementary material of [LZC+18].



(a) (b)

Figure 2.6: (a) Boundary singular vertex intersected with a yellow hemisphere. The green boundary of the hemisphere corresponds to intersection with the hex mesh boundary. (b) Triangulation of the hemisphere, with Vertices correspond to intersections between hex mesh edges and the hemisphere.

2.1.5 Structure Regularity

As in quad meshes, the *regularity* is introduced as a measurement of the topological quality of hex meshes [BLP+13, PCS+22]. We can classify hex meshes into four categories based on the regularity of their topological structure. The mesh regularity is strongly related to the singularity graph and the *base complex* constructed from it. The *base complex* we refer to here is the coarsest block decomposition. It can be obtained by tracing the facets at singular arcs until it terminates at the boundary or other singular arcs. Similar to the quad mesh case, the classification is according to the ratio r_V between the number of singular vertices n_s and the amount of all vertices n_v, and the ratio r_B between the number of blocks n_b and the number of all cells n_c.

1. *Regular* hex meshes are essentially voxelized cubes deformed to approximate geometric shapes. All mesh vertices are regular except for those on the 12 edges of the abstract cube. In practical applications, the simple structure of regular meshes is favorable as the data storing and connectivity querying are straightforward. However, the highly limited topology often results in geometrically poor hex meshes on complex objects and thus are barely usable.

2. *Semi-regular* hex meshes can be seen as a few regular blocks conformally stitched. Only boundary arcs of blocks can be singular arcs of the hex mesh. The value of r_V and r_B are typically low in this class. Owing to singularities, much more complex shapes can be represented as *semi-regular* hex meshes with lower geometric distortion. This category is preferred in many industrial applications since a few blocks allow for designing customized data structures for efficient storage and fast query. However, manual effort is required to generate *semi-regular* hex meshes with semi-automatic hex meshing tools, e.g., part of many industrial software products.

3. *Valence semi-regular* hex meshes have a similar number of singularities but a more intricate base complex structure than *semi-regular* hex meshes. For a fixed singularity graph, the length of singular arcs w.r.t. the number of hexes determines the structure of the *base complex*. Improper length results in a more complex block decomposition. Thus, the value in r_V is comparable, but r_B is higher than *semi-regular* hex meshes. This class of hex meshes can often be generated by automatic hex meshing algorithms, such as field based and polycube based algorithms.

4. *Irregular* hex meshes contain a large number of singularities. r_V and r_B are much higher than the other three classes of meshes. Therefore, it is not well-suited for applications that would benefit from the coarse block structure of hex meshes. Such meshes are often produced by grid-based hex meshing algorithms when the domain boundary is not aligned with canonical axes, thus creating singularities.

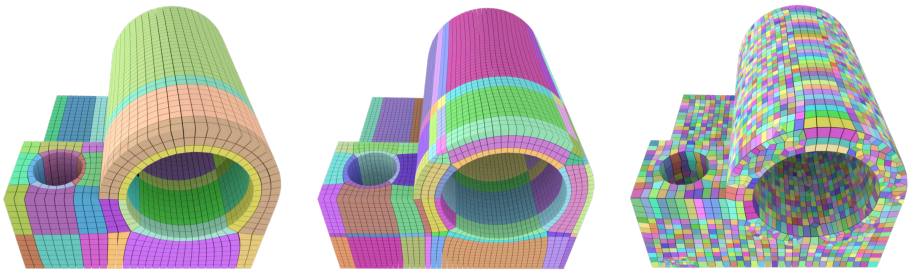


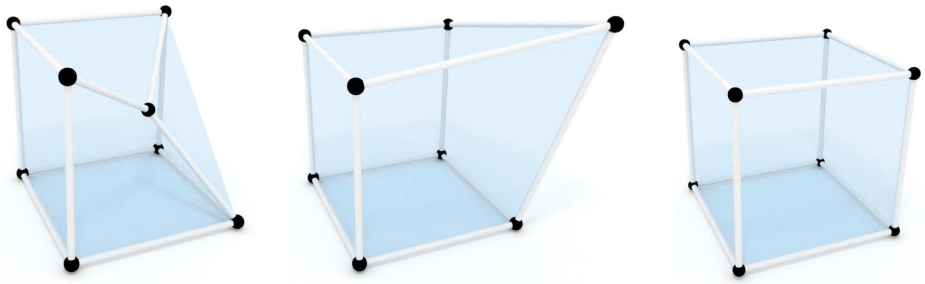
Figure 2.7: Left: the block decomposition of a hex mesh generated with frame field based method. Middle: the block decomposition of a hex mesh generated with polycube based method. Right: the result with octree based method. Frame field and polycube based methods often produce valence semi-regular hex meshes, which might contain fewer blocks depending on the quantization [CLS16, BBC22]. Octree based method generates an irregular hex mesh.

Note that it is a loose classification w.r.t. the topological complexity of hex meshes. There is no exact boundary among the semi-regular, valence semi-regular, and irregular hex meshes. Importantly, this measurement is only meaningful with fixed target complexity. Otherwise, refinement of *irregular* hex meshes can easily convert them to *semi-regular*. Moreover, structure regularity is coupled with a geometric quality. It is not necessarily true that a mesh with coarser block decomposition has a better geometric quality. An extreme example is that a large distortion of mesh elements is inevitable in mapping regular grids to an airplane. Therefore, hex meshing is a multi-objective problem, and there is no simple answer to what a good hex mesh is. Nevertheless, frame field based algorithm tends to produce singularity graphs that naturally arise, resulting in hex meshes with a good combination of topological and geometric quality. Figure 2.7 shows different hex meshing results of the joint model with frame field based method, polycube based method [GLYL20] and octree based method [GSP19] respectively.

2.2 Hexahedral Mesh Geometry

The geometric quality of hex meshes significantly impacts simulations in terms of the efficiency of the computation and accuracy of the results. Although measures of the geometric quality of hex meshes depend on applications, *element quality*, *fidelity*, and *feature preservation* are aspects often considered.

Element quality refers to the quality of each cell in hex meshes. Geometrically, a hexahedron is a trilinear hexahedral element specified by a map $f : [0, 1]^3 \to \mathbb{R}^3$. The measurement of the element quality depends on specific applications. Elements with a perfect cube shape are preferred in many finite element problems. In contrast, in fluid simulations, anisotropic elements (compressed in one direction) are desired at the boundary layers to capture the flow features. Different metrics have been proposed for hex meshes, such as scaled Jacobian, edge ratio, and skew. We refer to [SEK+07] for a detailed summary. As pointed out, depending on the specific application, one or another might be of major concern. [GHX+17] develops a frame work to analyze the correlations of various quality metrics. One frequently used metric in practice is the Jacobian determinant of the map $det(J_f)$ [Knu01]. The minimum Jacobian of every point in the element is of particular interest because inverted elements with negative minimum Jacobian can lead to inaccurate results or failure to converge in computation. Fig. 2.8 shows the cube with the Scaled Jacobian of negative, positive value and 1.0, respectively. In practice, post-processing is often applied after mesh generation to untangle invalid elements and improve the geometric quality. Various post-processing algorithms are developed for this purpose which we will review in Chapter 3.



(a) (b) (c)

Figure 2.8: Scaled Jacobian measured at corners of hexahedra. (a) Negative Scaled Jacobian. (b) Positive Scaled Jacobian. (c) Scaled Jacobian is 1.

Fidelity describes whether the mesh faithfully represents an input geometric shape. Accurate representation is essential in simulation to achieve a realistic result, especially when physical behavior at small details and key features are of concern. A commonly used metric is the so-called *Hausdorff distance* [KYKK19], which measures the distance between the surface of the input geometry and the surface of the output hex mesh.

Feature preservation is another crucial point that should be taken into account by practical hex meshing algorithms. In practice, feature points, arcs and patches often come along with the input geometry, and the output hex mesh should preserve them. There are both geometric and topological meanings in feature preservation. Geometrically, feature entities of the output hex

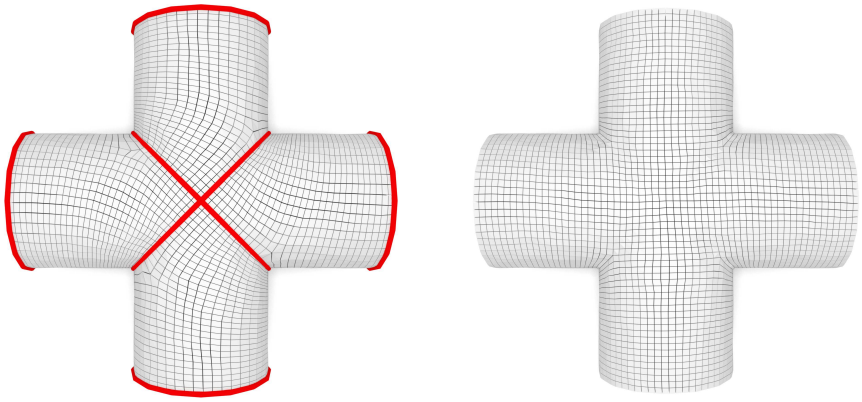


Figure 2.9: Hex meshing results of a cross-cylinder. Left: the hex mesh result, which respects the embedded features. Right: the hex mesh without considering features. No simple post-processing is capable of recovering the features.

mesh should be located at features of the input geometry. Topologically, feature nodes should be represented as hex mesh vertices; input feature arcs must be meshed as chains of hex mesh edges; feature surfaces should appear as patches of hex mesh faces. Feature preservation is essential in some applications because imprecision at features where boundary conditions are applied would strongly influence the solution of the PDE, such as in the aerodynamic simulation. In applications, for instance, multi-material optimization, interior feature preservation is also required when features exist inside the geometric volume. While the quality in other geometric metrics can be improved by post-processing, feature preservation must be considered in the mesh generation step. Simply projecting mesh vertices to features without changing the connectivity could result in degenerate or inverted elements (c.f. Fig. 2.9).

A hex meshing algorithm that can provide good quality in both topology and geometry is the ultimate goal. However, it is challenging because of its multi-objective nature. Topological simplicity and geometric quality are sometimes contradictory, e.g., over-simplified topology may result in highly distorted elements. A reasonable compromise must depend on the specific application. Besides, different element quality metrics may have distinct requirements for generated meshes. In this thesis, we mainly focus on the frame field based hex meshing algorithm where the hex mesh topology in terms of the singularity graph is of our primary concern, and feature preservation is also taken into consideration. Other geometric quality metrics are left to the post-processing procedure.

Before presenting the fundamentals regarding frame field based hex meshing methods, we review the general hex mesh generation and processing algorithms to have an overview of the state-of-the-art methods in this community. With the prerequisite knowledge on hex meshes, we can proceed to Chapter 3.

Chapter 3

Hex Meshing and Post-processing Methods

Research on hex mesh generation and processing have been ongoing for decades. Significant theoretical and practical progress has been made, and in particular, the computer graphics community has contributed substantially in recent years. Extensively reviewing all developed techniques is out of the scope of this thesis. It will shortly summarize related methods and briefly compare them in this chapter. For readers interested in more details, we refer them to the corresponding publications or surveys [AFTR15, YZL15]. Especially the recent comprehensive report [PCS+22] covers the most important approaches and provides a broad perspective on hex meshing and processing.

Generating satisfactory hex meshes is a difficult task. Approaches attempt to tackle this problem from different perspectives, e.g., automatic or semi-automatic ways, directly decomposing volume into grids or mapping-based methods, and primal or dual approaches. Post-processing techniques have been developed to obtain hex meshes with better geometric quality and topological structure, such as untangling inverted elements [LSVT15, XGC18], structural enhancements [CAS+19, GDC15]. A combined framework that generates hex meshes and then applies post-processing is sometimes necessary to achieve practically acceptable hex meshes.

3.1 Advancing/Receding front

One classical approach that targets an automatic generation of hex meshes is the advancing front method. It starts from the boundary of a shape and creates a layer of hex elements, advancing towards the interior of the domain layer by layer until the collision is detected. The last layer of elements either fills the volume or an interior void remains. Two variants of advancing front methods exist depending on whether the propagation starts with a prescribed quad mesh of the boundary or not, the constrained and the unconstrained. Fig. 3.1 (a)-(c)shows the general idea of a constrained approach in 2D. Iteratively inserting layers of hex elements to the fronts is straightforward, while the remaining small inner void can be tricky to mesh. The geometrical quality of hexes can be pretty low, especially if the domain is complex. More importantly, [Mit96] proposed

the necessary topological condition of the existence of a hexahedralization: any hex mesh of genus zero exhibits an even number of quadrilaterals on its boundary. Therefore, it is impossible to fill the inner void with hex elements if an odd number of quads surround a vacancy. In addition, [Eri13] generalizes the condition to arbitrary genus that the number of quad is even and there is no odd cycle in the quad mesh bounding the surface of the hex mesh.

The 2D illustration of unconstrained methods [SOB05, SKOB06, SKO$^+$10] is in Fig. 3.1 (d)-(f). Instead of explicitly generating layers of hex elements, it starts from the boundary surface of the geometry and progressively decomposes the domain into layers before the fronts collide. The inner void is usually meshed with simplicial elements, e.g., tetrahedra, and then split into hex elements. Since the boundary of the domain is not constrained by any quad mesh, it is possible to create hex elements from the boundary of the inner void and propagate them to the boundary through the generated layers. However, the mesh quality of the inner void can be rather low, and it determines the quality of the boundary meshes. The referred receding front methods [RRGS10, RGRS12] attempt to

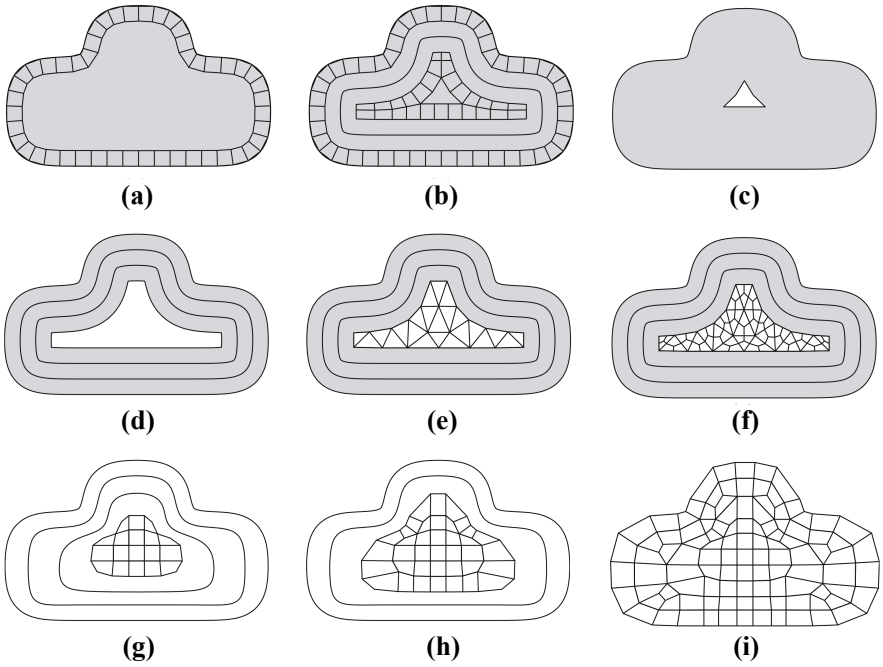


Figure 3.1: Constrained: (a) the first front, (b) the last front and contours of the previous fronts and (c) unmeshed void; Unconstrained: (d) fronts and final void, (e) simplicial mesh of the void, and (f) splitting simplicial mesh; Receding front: (g) Combined level sets and the first front, (h) intermediate layers and (i) final layer. Image from [RGRS12].

combine the advancing front methods, which usually produce high-quality hex meshes near the boundary, and the grid-based methods that generate high-quality meshes in the interior. They pre-compute two level sets : one from the domain's boundary and the other from the inner seed, which is templated. Then they blend the level sets and generate hex meshes from the inner layers to the

outer layers. The receding front can generate hex meshes with better quality than previous works. However, it can only be applied to models with ball topology. It is also not as reliable as the others to generate high quality meshes for general shapes, specifically when models vary dramatically in thickness.

3.2 Volume Decomposition

Another category for hex meshing is volume decomposition. Generally, a domain is subdivided into components with simple topological structures that can be easily meshed. Different strategies have been proposed in this direction, such as sweeping, block decomposition, and medial-axis based decomposition.

Because the extrusion operation is extensively involved in generating CAD models, "sweeping" methods are a natural choice for hex meshing. Early versions [SS96, Knu98, LG97] are applied to volumes in which one source and one target surface are identified, classified as one-to-one methods. Quad meshes are generated on the source surface and swept through the volume to the target surface for structure hex meshes. Unfortunately, most CAD models have more than one source and one target in reality. Algorithms that automatically break down extrusion geometry into these simpler components have been developed. Extrusion geometries with several source surfaces but only one target surface can be handled by many-to-one algorithms [SBO06]. These algorithms discretize each sub-volume separately after decomposing the initial volume into one-to-one sub-volumes. A step forward, several algorithms [LBW00, RGRS11, WGWC18] for meshing many-to-many extrusion geometries with multiple sources and target surfaces were proposed (see Fig.3.2). These methods are relatively mature and used as workhorse algorithms in many commercial software products [ANS22, Alt22, CUB22]. Generally, manual decompositions of CAD models into sweepable sub-domain with user effort is required in the first step, and then the automatic sweeping is performed subsequently.

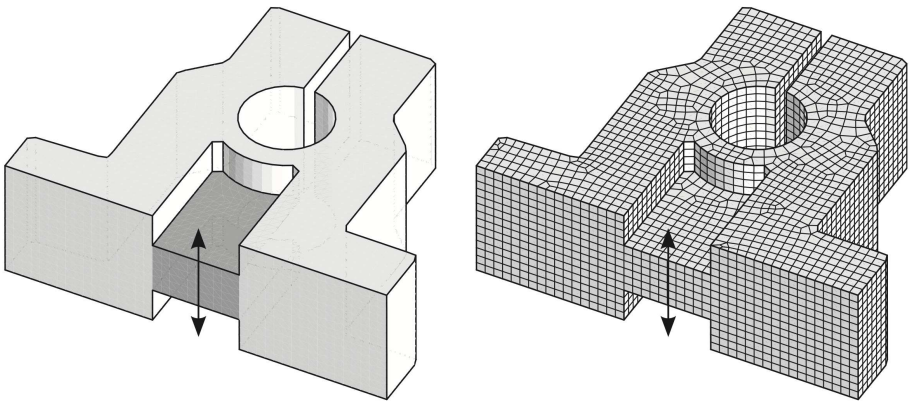


Figure 3.2: Sweeping: (a) Mechanical piece decomposition into two sub-volumes. (b) Mesh generated using the multi-sweeping method. Image from [RGRS11].

Some approaches investigate in the direction of automatically decomposing volumetric domain into blocks. The authors of [KLSO12] proposed an algorithm that adopts the notion of the fundamental sheet introduced in [LS10]. A CAD

model is first tetrahedralized and converted to a hex mesh via THex template [Car02]. Then fundamental sheets are inserted into the mesh. By carefully extracting non-fundamental sheets in an iterative manner, they can generate hex meshes having only 3- or 4-valent vertices. [WSC+17] introduces curve-related sheets to capture the boundary entities, and the validity of topology and geometry is claimed. [KLF14] proposes a volume decomposition approach by exploiting the singularity structure of frame fields. A few successful examples are shown in the paper. However, the issue of this method is strongly related to the fundamental problem that we will discuss in detail in Chapter 4: the singularity structure of a frame field does not necessarily exhibit a meshable structure. Given a surface mesh and 2D frame field, [LPP+20] automatically generates field-aware cutting loops for a block decomposition. It delivers feature-preserved pure hex meshes on many models upon success; otherwise, hex-dominant meshes. However, it is not suitable for models with interior features.

The medial axis has shown the capability of guiding volume decomposition and generating high quality hex meshes for specific shapes. [ZBG+07] designs an algorithm for vascular shapes. A curve skeleton is used to approximate the vessel structure. The tubular sub-domain is easy to mesh via sweeping methods, and the junctions of the skeleton are handled with a set of templates. Inspired by the work of [ULP+15] where a curve-skeleton is adopted to derive a quadrilateral base complex of a triangulated surface, [LMPS16] extends it for hex meshing (refer to Fig.3.3). A volumetric subdivision scheme is provided to adapt the topology of the mesh to the local thickness of the tubes. For more general shapes, [PAS95, PA97] use the medial surface to subdivide the volume to simple primitives, and then the sub-domains can be hex meshed with a midpoint subdivision scheme [LMA95]. However, it has trouble handling objects with N-valent vertices (object vertices that have more than three incident edges), and the geometric quality of the hex meshes is often poor.

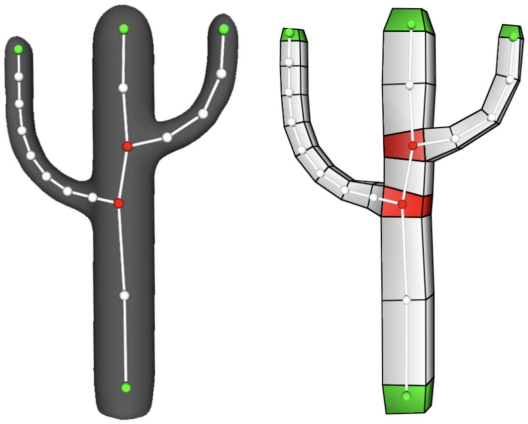


Figure 3.3: Derive from the curve-skeleton of a triangle mesh (left) a volumetric decomposition in tubes (white), branching cubes (red), and terminal cubes (green). Each element in the tubular structure is a hexahedron. Image from [LMPS16].

3.3 Grid based methods

Grid-based approaches follow the intuition that any geometric domain can be approximated with Cartesian grids. The basic idea is to create an initial voxel mesh, and then different strategies are employed to deal with the mismatch between the initial mesh's boundary and the shape's boundary. It is the only class of hex meshing methods that can automatically and reliably generate all-hex meshes for arbitrary input geometry. Implementations of grid-based methods are available in professional software [Cor22, CUB22]. Although grid-based approaches outperform others in terms of robustness, the generated meshes are not competitive because of several intrinsic issues of the methods. Firstly, since the grid is fixed, the results are orientation-dependent. Rotating a shape will lead to a mesh with a different global structure. Additionally, adaptive grids are essential for the geometric fidelity w.r.t. small features while having a reasonable mesh size. The transition from a denser to a coarser region can create an over-complicated singularity structure. Moreover, an adaptation of the meshes to conform to the surface of the domain also increases the irregularity. Hence, the meshes are highly unstructured and far from a coarse block decomposition. Although, a computationally expensive structure simplification algorithm [GPW+17] might be helpful.

[Sch96] is one of the first works which introduced the grid-based approach. A regular lattice is used to create the initial mesh, and the boundary region is meshed by constructing an isomorphic layer to the initial mesh's surface. [ZZM07] uses the density maps, which are constructed according to the surface curvature and local feature thickness to guide the adaptivity. An inside-out strategy is used to project the boundary vertices of the core mesh to the surface boundary. Inserting and collapsing techniques are employed to improve the geometric quality. To support the multi-domain, [SLK04] uses an outside-in and inside-in hybrid of the grid-based method.

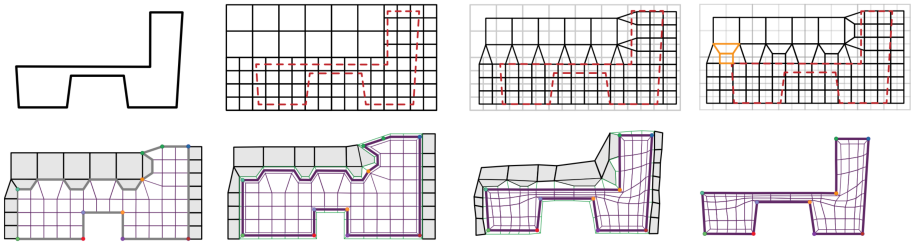


Figure 3.4: 2D illustration of the feature preserving octree-based hex-meshing. Top row: adaptive quadtree constructed from the input, dual of the quadtree, quad mesh, and scaffold mesh after splitting all cells. Bottom row: topological matching of features, padding of both the target mesh and the scaffold, mesh deformation to fit the input, and the pure output quad mesh. Image from [GSP19].

Other variants are the octree-based approaches [ZB06, ISS09, Mar09, EPOM11, ZLX13, EE14, OSE17] which highly reduce the size of the mesh. Due to the non-conformity between octants, hanging vertices are left in the final meshes, and different templates are proposed for conforming transitions. [Mar09, Mar16]

design an approach that can robustly produce pure, conformal, and valid meshes while preserving the sharp angles ($\geq 30\,^\circ$). [GSP19] proposes a new pipeline that behaves well in terms of element validity and geometry fidelity: small geometric features and sharp features in CAD models can be preserved; a user-specified distance deviation from the input surface is ensured; avoidance of global intersections is guaranteed. The specific steps of this algorithm are described in 3.4. However, this method is highly demanding regarding computation time and memory use, making it exorbitant for practical applications. [LPC21] enumerates all possible configurations of the transitions from a dual perspective and devises an optimal scheme that yields hex meshes with the simplest singularity structure and a considerable reduction in the element size count. The authors of [PLC+21] generalize the pairing criterion and solve a sequence of linear problems to reduce both grid and mesh size.

3.4 Polycube Methods

Polycube methods, which have been introduced by [THCM04] for texture mapping, have also been used to generate hex meshes [HXH10]. A polycube is a volumetric object whose surface normal is aligned to one of the principle axes $(\pm x, \pm y, \pm z)$. The tetrahedral mesh of a shape is volumetrically mapped to a polycube, called the polycube map (a formal definition is in Chapter 4.3). Then the map can be subdivided into hexahedra with a regular lattice. Finally, an inverse map is used to compute the position of the hex mesh vertices in the original domain. Fig. 3.5 shows the polycube map of an elephant model and the hex mesh result. A polycube is often generated with a specially designed energy involving two terms. One term iteratively deforms the surface triangles until their normals align to the global axes, and the other term penalizes large distortion and inverted elements.

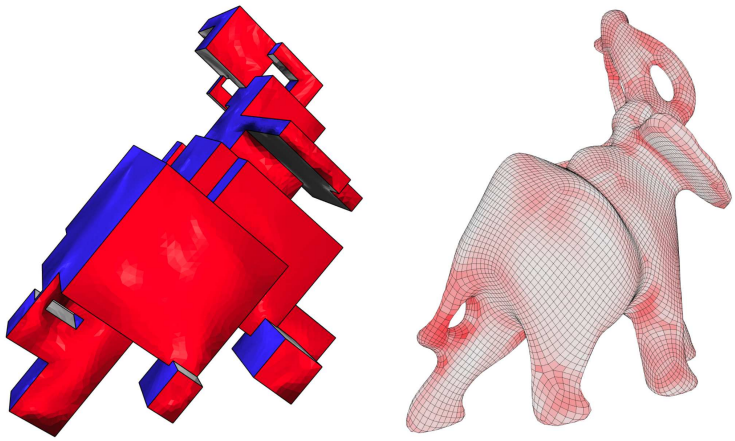


Figure 3.5: Left: the volumetric polycube map. Right: the hex mesh result. Image from [FBL16].

Various methods have been developed to deform volumetric objects into polycubes. [GSZ11] uses a rotation-driven deformation to softly align most

surfaces of the object to the principal axes. Then, a position-driven deformation with hard constraints is followed for a strictly aligned polycube. [LVS+13] formulates the segmentation as a multi-label graph cut problem to reduce the deformation distortion while reducing the number of singularities. The results of the two methods are sensitive to the orientation of the input mesh. Instead of a pre-computed polycube segmentation, [HJS+14] minimizes the l_1 norm of the surface normals to obtain a naturally arisen polycube structure. A user-guided control of the polycube shape is also supported. In the follow-up work [FXBH16], the frame field is used as guidance in the previous l_1 approach, and cuts are considered to support the polycube construction of higher genus objects. However, the computation is expensive due to the nonlinear optimization of the l_1 norm. A more efficient approach proposed by [FBL16] interleaves the normal smoothing and the normal alignment deformation, and the reference labels are updated after each iteration. Although an evaluation on a medium sized dataset is reported in [FBL16], there is no guarantee that a globally consistent polycube can be automatically achieved. User-involved post-processing is necessary to remove artifacts, such as faces with less than four edges. [SR15] observes some challenging cases where the slight normal deviation from the principle axes may lead to the wrong alignment of the polycube surface, as shown in Fig. 3.6. Interactive approaches for polycube construction are also developed.[YWL+20, LZS+21]

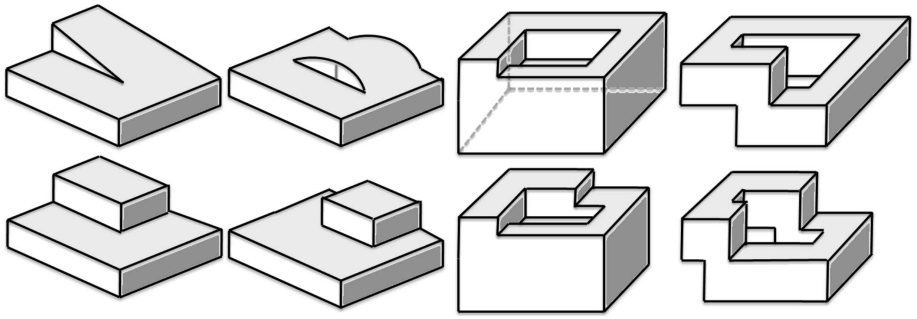


Figure 3.6: Top: no solution can be found with all normal constraints satisfied. Bottom: fixing normal constraints by adding steps. Image from [SR15].

The quality of the polycube map has a high impact on the hex meshing results. A polycube that is invert-element free and of low distortion is often desired for hex meshing purposes. In [GSZ11], deformation gradients computed according to the minimal rotation from surface normals to the closest global axes are used for minimizing the distortion in deformation. An as-rigid-as-possible deformation is employed in [HJS+14] to reduce the distortion. Note that these energies do not penalize the inversion sufficiently such that locally non-injective mapping can be produced, especially in concave regions. [FBL16] includes the AMIPS energy in the formulation, which goes to infinity with the presence of degenerate or flipped elements. Similar techniques [RPPSH17, DAZ+20, GKK+21] can be adopted in this setting. Although local injectivity is ensured, the deformation space is also reduced such that no solution which satisfies all axis-aligning constraints might be found [FSZ+21].

Unlike frame field based hex meshing, polycube methods restrict all singu-

larities to the boundary. This implicit property can induce significant distortion on the map. A naive way that most approaches employ is to add a buffer layer of hex elements globally. [CAS+19] designs a selective padding scheme by solving a binary problem that considers the quality improvement, increase of complexity, and the number of singularities. To further improve the distortion of the polycube map, [GLYL20] iteratively inserts interior singularities by diagonally cutting the polycube open through selected polycube edges where the distortion is high (see Fig.3.7). However, it only introduces limited types of interior singularities.

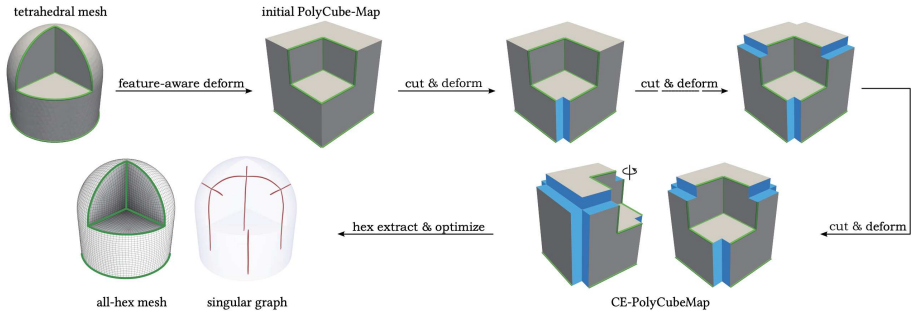


Figure 3.7: The algorithm pipeline of the CE-PolyCubeMap method. Image from [GLYL20].

Since it is highly practically relevant, feature preservation is one of the main challenges that polycube methods need to face. The only work that explicitly considers feature alignment is [GLYL20]. They develop a polyline deformation algorithm to cast out the feature edges which can not be kept in the polycube maps. Then they deform the input shape to a polycube with the guidance of the remaining feature edges. Input feature edges can only be partially preserved and represented as hex mesh edges in this heuristic approach. It is because feature edges must align to the isolines in polycube space. Because of the structural limitation inherited to polycube methods, only a subset of feature configurations can be preserved, which is far from sufficient in practice, e.g., more than three incident feature edges at a convex region. Moreover, to our knowledge, no method so far can preserve features that exist in the interior of the input shape. On the contrary, the novel frame field based approach developed in Chapter 7 ensures that all vertices are locally meshable. Upon success, all feature edges can be correctly represented as hex mesh edges, including the interior features.

3.5 Post-processing

After hex mesh generation, post-processing can improve the quality of a hex mesh, including both the topological and geometric quality.

Base complexes can be used to progressively simplify hex mesh topology through a series of operations [GDC15, GPW+17]. The key idea is to reduce the complexity of structure with simplification operations of collapsing sheets and chords of the *base complex* in the input hex meshes (refer to Fig.3.8). [XLZ+21] designs an improved ranking scheme of the simplification operations and achieves better results in complex hex meshes. These algorithms claime to

automatically and robustly produce coarse hex meshes when paired with octree methods. However, time efficiency is often limited and the simplification tends to get stuck in the local minimal.

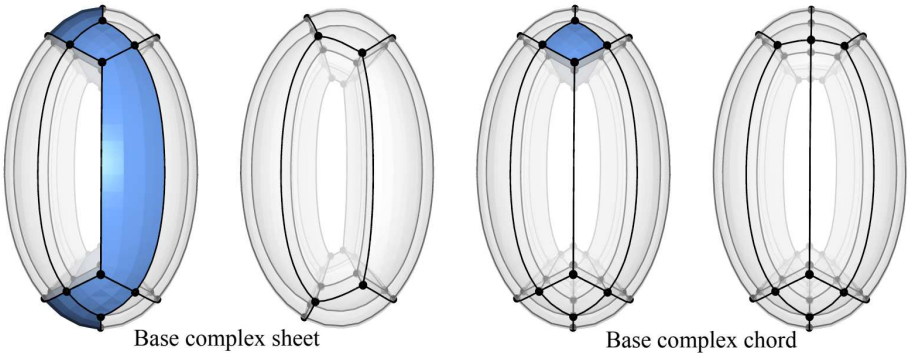


Figure 3.8: Collapsing a base complex sheet or removing a chord on the global structure of a hex mesh can reduce the number of blocks. Image from [GPW+17].

The immediate output of hex meshing methods often contain poorly-shaped elements or even inverted elements with negative determinants of the Jacobian matrix. Post-processing through geometric optimization improves the shape of element, mainly to obtain inversion-free hex meshes. Most optimization methods [Knu03, WSRRR+12, RGRS14] focus on the quality improvement of the Jacobian determinants at the eight corners of each hex element. The work of Livesu et al. [LSVT15] performs iterative local improvements on the corners of the hex elements, improving the scaled Jacobians. With similar concepts, [XGC18] first untangles the inverted region by improving corners with a relaxation of the surface preservation and then deforms the volume to approximate the surface while keeping the validity of elements. The widely used library Mesquite [BDK+03] implements many quality metrics and optimization algorithms. The suggested usage is to untangle hex meshes first with [Knu01] and then improve the element shape. However, there is no guarantee that the output hex meshes are inversion-free. [JWR17, MPZS20] take the Jacobian determinant of every point in the hex element into account, which provides a sufficient condition for the validity of a hex element.

Above are brief reviews of general hex meshing and processing algorithms.Frame field based hex meshing techniques and all related works will be discussed in Chapter 4.

Chapter 4

Frame Field Based Hex Meshing

A frame field can be viewed as a relaxation of the differential of integer-grid maps from the tet mesh to the parametric domain. Similar to hex mesh topology, the topology of a frame field is also characterized by its singularity graph. The set of frame field singularity graphs is a superset of the hex mesh singularity graphs, meaning a corresponding frame field exists for any given hex mesh. In other words, a valid integer-grid map is possible if the singularity graph of the frame field is in the set of hex mesh singularity graphs. This relation motivates the frame field based hex meshing methods that use the frame field as guidance in seeking valid IGMs. Unlike other hex meshing methods, imposing various constraints on the hex mesh topology (e.g. the singularity graphs via polycube methods are restricted to the boundary), frame field based hex meshing exhibits high flexibility in producing hex meshes of arbitrary topology. Therefore, it is one of the most promising approaches toward the "holy grail" of an automatic hex meshing algorithm for general shapes.

In this chapter, we cover the fundamentals of frame field based hex meshing techniques. We begin by reviewing state-of-the-art related methods for quad meshing of surfaces before moving on to related approaches for hex meshing of volumes. Along the way, we highlight some subtle challenges in this generalization to the volumetric setting.

4.1 Surface Approaches and Quad Meshing

Field based parametrization approaches to quad remeshing have been extremely successful in producing high quality quad meshes. In these approaches, a 2D frame field is used to guide a parametrization of the surface in question to a quantized cone manifold. This allows for a quad mesh to be pulled back onto the surface. We cover some relevant works below and refer to surveys on directional field design [VCD+16] and quad remeshing [BLP+13] for further details.

One of the major contributions in this thesis, which reconstructs 3D frame fields with prescribed singularity graphs, is heavily influenced by the work of Ray et al. [RVLL08], who produced N-symmetry direction fields with input indices satisfying the Hopf-Poincaré formula. This was achieved by "zippering",

or specification of topological matchings in a smart sequential order, which guarantees correct vertex indices. The "chart-zippering" portion of the algorithm described in Chapter 5 can be understood as a volumetric extension of this strategy with the additional challenge of connecting singularities correctly. Related work by Crane et al. [CDS10] achieved a similar goal by directly optimizing the smoothest (continuous) connection rather than initially constructing (discrete) topological matchings. The globally optimal smooth fields of [KCPS13] inspire our method for generating the frame field with fixed topological matchings, described in Chapter 5. In 2019, the authors of [CC19] generalize the idea of [CDS10] to 3D and reconstruct the frame field of a given singularity graph by optimizing the *Darboux derivative*. The advantage of this formulation is that no discrete *matchings* are required for the optimization. However, as nicely-shaped tube geometry is required as the input, they only show a few results on simple models.

Full pipelines for quad meshing based on 2D frame fields are proposed, e.g., in [BZK09, BCE+13] and form the basis of modern quad meshing software. 2D frame field topology is fundamental for field-based quad meshing. In the 2D setting, [MPZ14] states that for a locally injective parameterization, the 2D frame field is required to align to feature curves, and additionally, there are no parabolic or elliptic sectors. This is enforced by adding constraints to sectors in their angle-based field representation. Moreover, they robustly construct a global parameterization by partitioning the surface into rectangular patches with field streamlines and then repairing zero loops via singularity insertion. Beyond field design, locally injective quantized parametrization [CBK15] and robust mesh extraction [EBCK13] techniques are required.

4.2 3D Frame Fields

A 3D frame field is a generalization to 3-dimensions, which represents the smoothly changing linear transformation in space. Locally a frame can be viewed as a parallelepiped, i.e., a linearly deformed (rotation, shearing, and scaling) unit cube. This parallelepiped can be explicitly defined as a set of six vectors $f = (\pm u, \pm v, \pm w), u, v, w \in \mathbb{R}^3$, which form the edge vectors of the parallelepiped. Note that three vectors u, v, w are linearly independent. In some literature, a subset of three vectors is used to denote a frame $f = (u, v, w)$. Given the reference frame, $\hat{f}$, which corresponds to a unit cube, a frame f is the reference frame linearly transformed by a 3×3 matrix $M : f = M\hat{f}$. When only the rotation information is of interest, a unit cube can be used to represent a frame, and in this case, u, v, w are unit vectors that are orthogonal to each other. As a unit cube is dual to an octahedron, it is also termed an octahedral field [SVB17, LZC+18]. In field based hex meshing, a frame field, as the guidance of the integer grid map, is the inverse of the Jacobian matrix of the map (further discussed in Sec. 4.3).

3D Frame Representation Many different ways have been used to express a frame in computation and visualization. A straightforward representation is a 3×3 matrix composed of the three linearly independent vectors u, v, w. The matrix $f = (u, v, w) \in \mathbb{R}^{3\times3}$ encodes not only the rotation but also the scaling and shearing, which supports the generation of adaptive and anisotropic hex

meshes [FHTB21]. Considering only the rotation, a common choice to express a unit orthogonal frame are Euler angles [HTWB11, RSL16, PBS20]. Unit quaternion [GPW+17, LZC+18] is a more compact representation that avoids gimbal locks. In this thesis, unit quaternions are used to represent octahedral frames. 3D matrices are adopted in the steps where shearing and scaling are necessary, for example, the parameterization and the optimization for integrable frame fields.

The representations mentioned above alone are not adequate for measuring the difference between frames because of the octahedral symmetry. In the matrix representation of the frame, there are 3! permutations of the axes and 2^3 sign combinations for 48 different configurations, forming the full binary octahedral group. 24 out of 48 combinations have positive determinants, called the chiral octahedral group O. Multiple rotations can represent the same octahedral frame. For instance, the frame $f_a = (u, v, w)$ rotated along the axis w by 90° results in $f_b = (-v, u, w)$. f_a and f_b are related by a chiral octahedral group transformation and thus represent the same frame, as permutation and sign of individual axis are irrelevant. With these explicit representations, the *matching matrices* [NRP11] are required for the smoothness measure between frames. Each *matching matrix* R is a permutation matrix that determines how two frames are combinatorially matched, and R is an element of O. These discrete variables in the frame field optimization significantly complicate the problem.

To overcome the non-uniqueness issue, the pioneering work [HTWB11] introduces a representation of frame fields using spherical harmonic coefficients, which is invariant to octahedral symmetries and thus enables the optimization of smooth boundary-aligned fields. A unique descriptor with octahedral symmetry $f_{[R]}(s) = h(R^T s)$ is designed to express the rotation of the frame via the rotation of the sphere, where $h(t) = t_x^2 t_y^2 + t_y^2 t_z^2 + t_z^2 t_x^2$ is a fourth order polynomial restricted to the unit sphere $S^2 = \{s \in \mathbb{R}^3 | \|s\| = 1\}$. Thus, the smoothness between two frames $R_{[a]}$ and $R_{[b]}$ is formulated as an integration over the unit sphere,

$$\int_{S^2} (f_{[R_a]}(s) - f_{[R_b]}(s))^2.$$

Projecting the descriptor onto a 4th-order spherical harmonics basis gives the representation of frames as $\mathbb{R}^9$ spherical harmonics coefficients. [CHRS18, GMS21] use 4th order symmetric tensor to represent the frame. [PBS20] further generalizes the representation to 15-dimension to enable scaling orthogonal frames along principle axes.

3D Frame Field Design Several methods aim at automatically generating feature-aligned smooth frame fields. The seminal work of [HTWB11] minimizes the *Dirichlet* energy of spherical harmonic coefficients in $\mathbb{R}^9$ and projects the coefficients back to the closest octahedral frame. [RSL16] formulates it as a least-square problem and directly enforces boundary alignments in the initialization. By applying the boundary element method, [SVB17] achieves frame fields of infinite resolution inside the volume while only a boundary triangulation is required. The work of [PBS20] proposes differential and algebraic descriptions of octahedral frames and applies manifold-based optimization (MBO) to obtain smooth frame fields. Singularity structure of better structure can be achieved

with the modified MBO scheme where the diffusion time is progressively decreased during the optimization. All the above methods take smoothness as the objective while the frame field topology is not considered, often leading to non-meshable frame fields.

3D Frame Field Singularities As analogous to N-symmetry vector fields on surfaces, the singularities of 3D frame fields can be measured by transporting the frame in an arbitrary chart at a point p along a closed loop in its vicinity. For the N-symmetry fields, the singular point type is described by the *index*, which is the angle difference between the frame and the transported frame divided by 2π. However, it does not naturally generalize to 3D frame fields[NRP11]. The singularity type is characterized by 24 elements of the chiral octahedral group. Instead of discrete singular points on surfaces, singularities of 3D frame fields exist as singular arcs that connect at nodes or end at the domain's boundary. Like hex meshes described in Chapter 2.1, frame field singularities also form singularity graphs.

Given a tetrahedralized domain, a typical sampling of frames is per cell, meaning each tet exhibits a unit quaternion q as the frame. Each tet is a chart c with a local coordinate system. The *matching* R_{ij} that specifies the connection between chart c_i and c_j is assigned to the halfface (dual halfedge) and the opposite halfface stores the inverse matching, as shown in Fig.4.1. The frame q_i from chart c_i expressed in chart c_j is thus $R_{ij}q_i$. The edge type of singularity can be derived from the matching. Note that the frame field *matching* here differs from the transition function defined for integer grid maps discussed in detail in Section 4.3.

Consider one-ring cells at an interior edge e, starting from the tet t_0 with frame q_0, and the dual cycle passes all incident tets $(t_0, t_1, t_2, ..., t_k, t_0)$ in counterclockwise order. The *edge type* of e with respect to tet t_0 can be checked via the concatenation of *matching* defined per halfface [NRP11, JHW$^+$14]:

$$type(e) = R_{k0}...R_{12}R_{01}.$$

This rotation product $type(e) \in O$ measures how the frame in t_0 is transformed along the dual cycle when it returns to the original tet t_0. There are three categories of $type(e)$ based on the conjugacy classes of the octahedral group [JHW$^+$14]. The edge e is regular if $type(e) = Id$. 9 different rotations around principle axes $(\pm u, \pm v, \pm w)$ by 90° or 180° results in *admissible* singular edges. The image of *admissible* singular edges in parametric space aligns to isolines and correspond to hex mesh edges. The other 14 rotation types are either along the face diagonal or cube diagonal, inducing the degeneracy of edges to points in parametric space. They are called *compound* or *complex* singular edges. Note that the edge type e is independent of the reference tet t_0 up to a local coordinate transformation. For the boundary edge type, similarly, we measure the rotation of the surface normals in parametric space. The surface normal n_0 corresponds to one of the axes A_{n_0} in the local coordinate system of tet t_0 and is transformed along an open dual path to the other boundary tet t_k via matching product R_e: $R_eA_{n_0}$. The rotation between $R_eA_{n_0}$ and the corresponding surface normal axis A_{n_k} in tet t_k shows the type of the boundary edge. A more detailed description is in Chapter 5.1

Singular edges of frame fields measured via *type* can only classify rotations modulo 2π. As for higher valence, the ambiguity of multiple 2π rotations makes

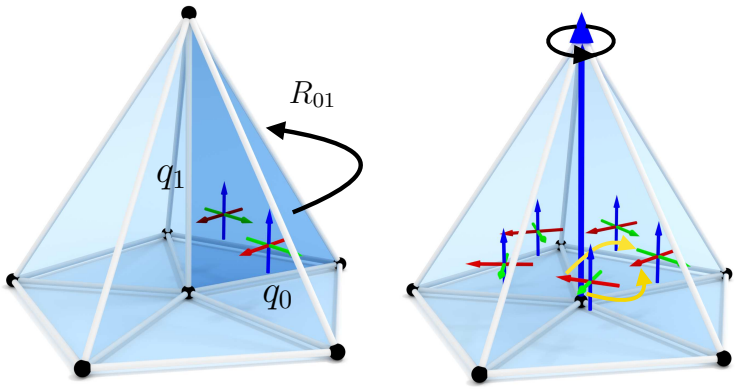


Figure 4.1: (Left) Frames in tet t_0 and t_1, and the matching R_{01}. (Right) Frame fields in the one-ring tets of a singular edge. Matched axes of two frames are changed after the circulation around the singular edge.

it impossible to distinguish. More precisely, we can check only valence three and valence five singular edges via matching concatenation. There is an ambiguity between valence two and valence six since rotations around u by $180°$ and $-180°$ result in the same frame. In organic shapes, singular edges of high valence rarely appear in an automatically generated smooth frame field. As discussed in Chapter 2.1, high valence hex mesh edges can be decomposed into valence three and valence five edges. However, in many practical applications, features explicitly encoded in CAD models can have feature vertices where multiple feature edges are joined. Higher valence types are necessary in this case since the features introduce hard constraints on the resulting hex meshes. To detect edge types of high valence, we accumulate the rotational angle difference α_{ij} between neighboring frames q_i and q_j in counter clockwise order. Assuming q_j and $q_i' = R_{ij}q_i$ are aligned to the axis u in chart c_j, the rotation is thus $R(u, \alpha_{ij}) = q_j q_i'^{-1}$. Subtracting the sum of angles $\sum_{f_{ij} \in F(e)} \alpha_{ij}$ with the dihedral angle $\alpha_d(e)$ at the edge e results in the angle defect $\alpha(e)$. The valence of the edge e is thus $\frac{\alpha(e)}{2\pi} + 4$. When $type(e) = Id$, meaning the rotation is $n \cdot 2\pi$, and there is no obvious rotation axis, a voting strategy needs to be employed to determine a consistent rotation axis in each local coordinate system.

Singular arcs meet at vertices, called singular nodes. Unlike singular nodes in hex meshes (refer to Chapter 2) with only a finite number of different configurations given restricted valences, the structure of singular nodes in frame fields is much more complicated. Even with bounded valence, there are infinitely many other arrangements of edges simply because one can attach arbitrarily many singular arcs to the singular node. Fig. 4.5 shows a typical invalid singular node type of frame field.

Frame field correction Of particular relevance to our work, two previous methods aim at automatically correcting the defects in frame fields [LLX+12, JHW+14]. The key idea of [LLX+12] is to remove the *compound* singular edges, which do not admit valid hex mesh edges by edge collapsing. An alternative approach [JHW+14] splits the compound singular edges into admissible singular edges. Both methods ignore the meshability at singular nodes, and consequently,

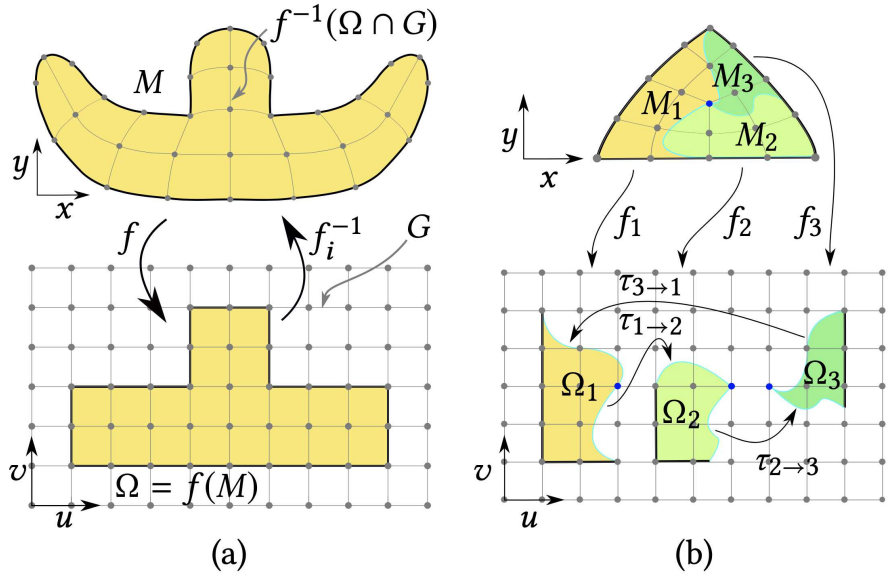


Figure 4.2: (a) Polycube map of volume M such that the mapped boundary aligns with the integer grid G and pulled back hexahedra tessellate M. (b) Integer-grid map with three boundary aligned charts $\{(M_1, f_1), (M_2, f_2), (M_3, f_3)\}$ that induce a singular vertex (blue). Transition functions τ define the parametric matching of grid directions.

only limited defects in frame fields can be repaired. Additionally, authors of [VSL16] discuss two reasons for non-meshability, singular edges deviating from the frame field directions and the existence of limit cycles. They reveal that it is ineffective to add constraints to frames to align to existing singularity graphs and then re-optimize frame fields. Furthermore, [RCR19] observes the typical non-meshable case of 3-5 singular curves, often seen in frame fields of CAD models with concave feature regions. They propose four ways to correct it, while only a heuristic that snaps singular arcs to the boundary is considered a feasible solution. Our novel algorithm described in Chapter 7 addresses all the local issues and achieves locally meshable frame fields.

4.3 Integer-Grid Maps

Polycube Maps One way to obtain a boundary-aligned hexahedral mesh is to deform the input region $M \subset \mathbb{R}^3$ with a map f that aligns the boundary ∂M with the grid of Cartesian integer isoplanes

$$G = \{(u, v, w)^T \in \mathbb{R}^3 | u \in \mathbb{Z} \text{ or } v \in \mathbb{Z} \text{ or } w \in \mathbb{Z}\}.$$

The decomposition of M into hexahedral cells is then obtained by $f^{-1}(\Omega \cap G)$, which pulls back those hexahedra from G which are covered by the image $\Omega = f(M)$, as illustrated in Fig. 4.2a. To guarantee a topologically valid hexahedral decomposition, not only boundary alignment but moreover, local injectivity of

f is required [BCE+13]:

$$(\textbf{Boundary Alignment}) \quad f(p) \in G \qquad \forall p \in \partial M \tag{4.1}$$

$$(\textbf{Local Injectivity}) \quad \det J_f > 0 \qquad \forall p \in M \tag{4.2}$$

with $J_f = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z] \in \mathbb{R}^{3\times 3}$ being the Jacobian matrix of f. Maps from this class are called *polycube maps* [THCM04]. Unfortunately, only hexahedral meshes without interior singularities can be generated. To achieve the class of all hexahedral meshes, polycube maps must be generalized to *integer-grid maps*.

Integer-Grid Maps Integer-grid maps similarly generalize polycube maps as manifolds generalize simple parametric representations. Imagine that the input region is equipped with an atlas of coordinate charts $A = \{(M_i, f_i)\}$, i.e. a partitioning $M = M_1 \cup M_2 \cup \ldots \cup M_k$ into k parts, each providing its own map $f_i : M_i \to \Omega_i$. To obtain a seamless hexahedral mesh, the piecewise polycube maps $f_i^{-1}(\Omega_i \cap G)$ need to be stitched at all points $p \in M_i \cap M_j$ contained in the intersection of charts. This can be done by restricting the transition functions $\tau_{i\to j} = f_j \circ f_i^{-1}$ between charts i and j to preserve the grid of integer isoplanes, i.e. $\tau(G) = G$. In [KNP07a, NRP11] it is shown that $\tau(a) = Ra + t$ where $R \in \mathrm{Oct}$ is one of 24 orientation-preserving octahedral permutations [SVB17] and $t \in \mathbb{Z}^3$ is an integer translation.

As illustrated in Fig. 4.2b, the use of charts and grid-conforming transition functions enable the stitching of hexahedral cells in a topologically irregular manner and in particular, provides the flexibility to create all types of singularities discussed in Chapter 2.1. Whether a point $p \in M$ is a singularity of the map can be measured by the holonomy of the connection induced by the transition functions. More precisely, if there is any cycle around p with a nonzero holonomy, p belongs to the singularities induced by the map f, denoted by $p \in S_f$. Alternatively, S_f is given by the locus of points where the differential is not well-defined. To guarantee that the conformingly stitched grid consists only of hexahedral cells, it is additionally required that all singularities $p \in S_f$ are mapped to the integer grid, leading to the following two additional conditions:

$$(\textbf{Conformity}) \quad \tau_{i\to j}(a) = R_{i\to j}a + t_{i\to j} \quad \forall a \in f_i^{-1}(M_i \cap M_j) \tag{4.3}$$

$$(\textbf{Singularities}) \quad f(p) \in G \qquad \forall p \in S_f \tag{4.4}$$

An atlas of charts $A = \{(M_i, f_i)\}$ where all f_i and $\tau_{i\to j}$ satisfy (boundary alignment), (local injectivity), (conformity), and (singularities) is called an *integer-grid map (IGM)*. Every IGM is guaranteed to induce a boundary-aligned and topology-preserving hexahedral decomposition of M. The reverse also holds: for each hexahedral decomposition, there exists a corresponding IGM.

Seamless Maps Without constraining $t_{i\to j} \in \mathbb{Z}^3$, the IGM is relaxed to the *seamless map* with $t_{i\to j} \in \mathbb{R}^3$. First, one solves the easier problem by dropping part of the constraints and then assigning integer constraints back. An early approach is to round $t_{i\to j}$ to the nearest integer and then resolve the problem with the fixed $t_{i\to j}$. Ideally, it can find a valid IGM with infinite resolution. However, it is rather fragile in practice and often has conflicts, leading to invalid IGMs. [BGMC22] (Volume Parametrization Quantization for Hexahedral

Meshing) extend the 2D quantization [CBK15] to 3D volume by introducing the 3D motorcycle complex. Given a valid seamless map ($\det J_f > 0$), it can robustly produce a valid IGM.

IGM Induced Frame Fields Consider one chart (M_0, f_0) of an IGM. Mapping the coordinate frame $[\hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{w}}]$ from the parametric domain Ω_0 to M_0 by means of the inverse differential results in a *frame field* $F = J_f^{-1} \in \mathbb{R}^{3\times3}$, which we abbreviate by $F = [d_u, d_v, d_w] \in \mathbb{R}^{3\times3}$. Geometrically, the frame field represents the local orientation of mapped hexahedral elements and is smooth within a single coordinate chart but potentially discontinuous across different charts due to the transition functions. The transformation rule for frames between neighboring charts M_i and M_j is induced by the transition function between Ω_i and Ω_j. It follows from the identity $df_j^{-1}(v) = df_i^{-1}(d\tau_{j\to i}(v))$, meaning that mapping a vector v from Ω_j to M is identical to first transitioning v to Ω_i and then mapping to M (cf. Fig. 4.2). Considering that $d\tau_{j\to i} = R_{j\to i}$, the resulting rule for transforming a frame F_i from chart i into representation F_j w.r.t. chart M_j consequently is

$$F_j = F_i R_{j\to i} \tag{4.5}$$

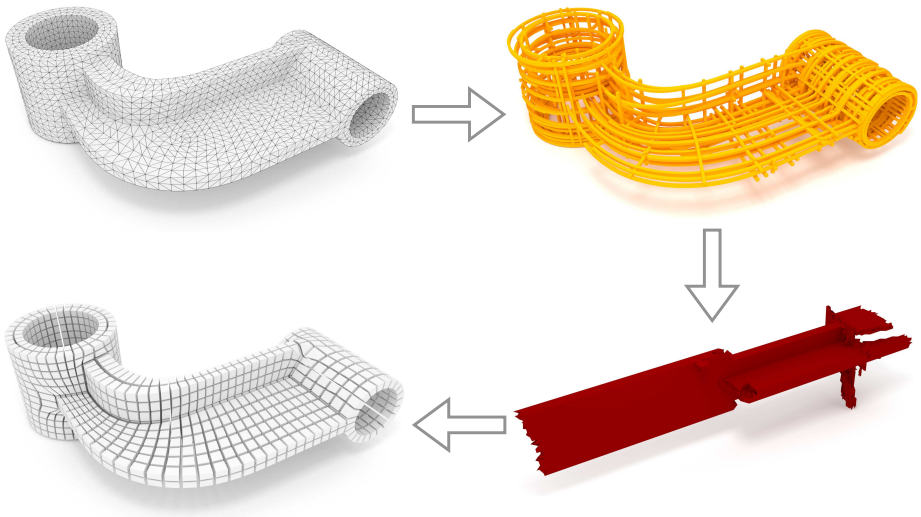
Note the inverse behavior compared to the transformation of a vector $v_i \in T_{\Omega_i}$ in the parametric domain following from Eq. (4.3):

$$v_j = R_{i\to j} v_i \tag{4.6}$$

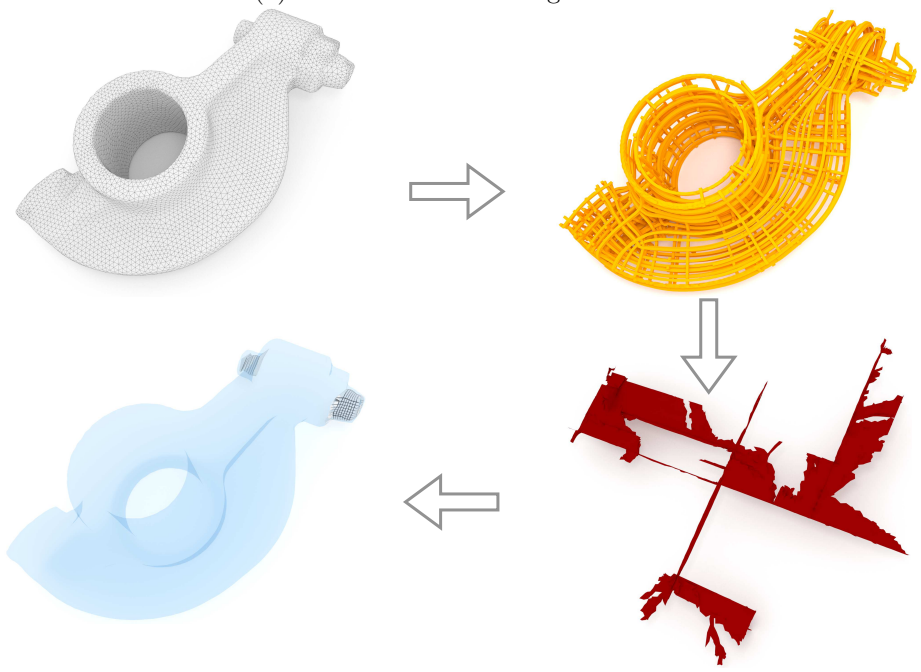
with $R_{i\to j}$ being the inverse of $R_{j\to i}$.

Octahedral Fields By means of the octahedral group Oct, a frame F extends to its axis set $\mathcal{A}(F) = \{\pm d_u, \pm d_v, \pm d_w\}$, which is a smooth field at all non-singular $p \in M \setminus S_f$, specifically across chart boundaries. This IGM-induced *octahedral field* is orthonormal in the metric of the parametrization. Conversely, a given octahedral field on M can be converted into a frame field by arbitrarily choosing a right-handed subset from the axis set for each chart (from here on frame/octahedral fields are used interchangeably). While every IGM induces an octahedral field, the converse is not true: not every field can be integrated into an IGM. If this is the case, we say the field is *globally hex-meshable*, or globally meshable for short, as the hex mesh given by the integrated IGM has the topology dictated by the field. In particular, any globally meshable field must have a singularity graph that satisfies the necessary conditions of Chapter 2.1.

IGMs for Hexahedral Meshing A straightforward hexahedral meshing approach optimizes for a low-distortion map in the class of integer-grid maps. Unfortunately, the resulting optimization problem is extremely challenging due to its high dimensionality, the strong non-convexity due to (4.2), and especially the huge number of discrete degrees of freedom due to (4.1), (4.3) and (4.4). Consequently, all known IGM-based hexahedral meshing algorithms perform a splitting approach [LLX+12, JHW+14], illustrated in Fig. 4.3. In the first step, a smooth and boundary-aligned octahedral field is generated, e.g. through [HTWB11, RSL16, SVB17], which is then used as a guidance field to find the most similar IGM [NRP11].



(a) Successful on the hanger model.



(b) Failed on the rockerarm model.

Figure 4.3: (a): splitting approach, an octahedral field (yellow) is generated to guide the IGM parametrization (red). (b): an invalid singularity graph induces constraints leading to a highly degenerate parametrization. Consequently, only very few hexahedra can be extracted [LBK16]

A fundamental problem with this approach is that the singularity graph of the smooth octahedral field is often topologically invalid [VSL16]. As a result, it is common that for seemingly high quality octahedral fields the IGM still heavily degenerates by violating condition (4.2), such that in most areas no consistent hexahedral cells can be extracted (cf. Fig. 4.3, bottom). Typical defects of the

singularity graph include local defects, which are singular nodes or arcs of invalid type or global defects.

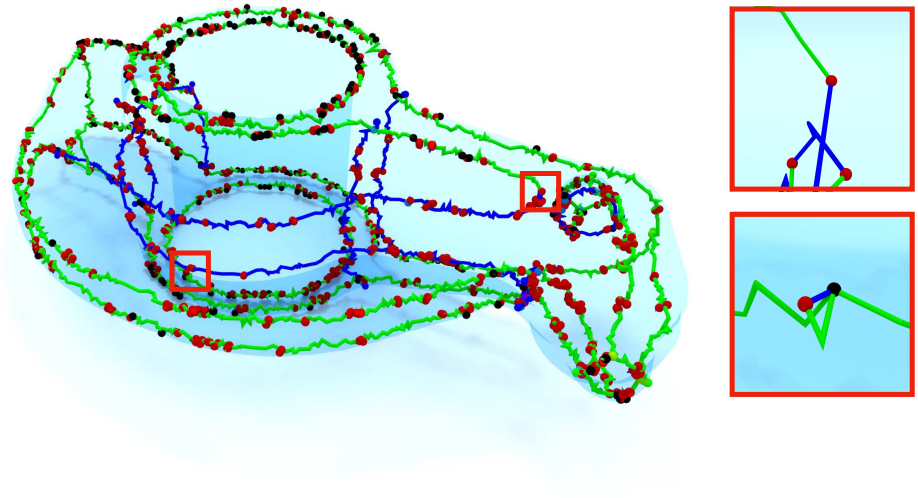


Figure 4.4: Typical singularity graph defects of smooth octahedral fields. Green and blue arcs are singularities of index $\frac{1}{4}$ and $-\frac{1}{4}$ respectively. Black spheres indicate invalid node topology, including cases where singular arcs touch the boundary tangentially. Red spheres denote the turning points.

Common invalid singular nodes that we observed are of type $(3, 0, 1)$ or $(1, 0, 1)$, which are turning points where a singular arc changes, e.g., from valence 3 to valence 5, and returns to its source. An example extracted from a smooth octahedral field is depicted in Fig. 4.5. Invalid arcs are often created when several arcs in the vicinity of a singular node snap on a common edge, creating a complex type. Such cases are often of local nature and can be resolved by methods like [LLX$^+$12, JHW$^+$14]. If an octahedral field has a singularity graph that is locally meshable, then we say the field itself is *locally hex-meshable*, or locally meshable for short.

Even if we have local meshability, the field may still fail to be globally meshable. For example, our global necessary condition stated in Chapter 2.1.4 might be violated. As another example, there might be no global meshing due to limit cycles (cf. [VSL16, SR15]). The analog of this problem occurs even in the surface case, for quad meshing [CZ17]. The example of a cross-field on a torus with two singularities of index $\pm 1/4$ is shown in this reference. An analogous octahedral field on a thickened torus will also lack global meshability. Another typical global defect that leads to non-meshable issues is the twist, as shown in Fig. 4.5. Although everywhere in the *bone* mesh is locally meshable, the twist of singular arcs causes a severe collapse in the parameterization. There is no known characterization of globally meshable fields or method to generate them. Automatically generating globally meshable fields is the ultimate puzzle in frame field based hex meshing and is left for future work.

Instead, this thesis first provides an alternative solution to the fully automatic approach. Assuming a globally meshable singularity graph, the algorithm

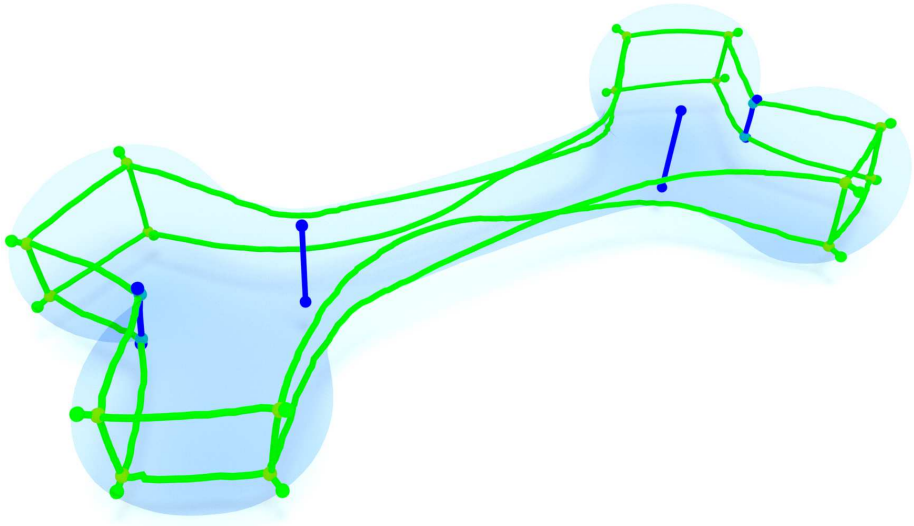


Figure 4.5: One twist of the singular arcs exists on each side of the bone mesh.

can generate the corresponding meshable frame field. Replacing the frame field in the splitting approach with this meshable frame field would result in a valid hex mesh with the same singularity structure. The algorithm is presented in Chapter 5. It then focuses on the automatic frame field based hex meshing pipeline, targeting improving the robustness. In collaboration with colleagues [BRK+22], we set up the HexMe dataset designed to test the robustness of hex meshing algorithms, particularly regarding the correct preservation of features. In Chapter 6, we introduce the HexMe dataset and evaluate the standard frame field based hex meshing pipeline, showing its poor behavior in terms of robustness on this dataset. As the major contribution, this thesis presents a novel algorithm that automatically repairs all locally non-meshable defects in Chapter 7. This novel hex meshing pipeline produces locally meshable frame fields, achieving significantly more valid IGMs on the HexMe dataset than the state-of-the-art methods [JHW+14, LLX+12].

Chapter 5

Singularity-Constrained Octahedral Fields

Given a tet mesh, the standard hex meshing procedure which often leads to invalid hex meshes is shown in the top row of Fig.6.1. This chapter advocates a modified splitting approach, including two additional steps: (i) repairing the singularity graph and (ii) generating a new octahedral field, where the corrected singularity graph is preserved. The bottom row of Fig.6.1 demonstrates the modified approach. The input singularity graph of a globally meshable field can be totally user-designed or manually repaired with the hint of a smooth frame field. The main contribution in this chapter is an algorithm for step (ii), which generates a smooth and boundary-aligned locally meshable octahedral field under the constraint of a prescribed singularity graph. Developing a general solution for step (i) is left for future work. Still, nevertheless, the set of necessary conditions in Chapter 2.1 is helpful on its own to support the correction of singularity graphs. In this regard, our algorithm can be used for the detection of invalid topology and provides information on where inconsistencies are located.

5.1 From Singularity Graph to Octahedral Field

Given a tet mesh $\mathcal{T} = (V, E, T, C)$ with vertices V, edges E, triangles T and cells C, and a singularity graph $\mathcal{S}$ embedded in the 1-skeleton $V \cup E$, our goal is to find a discrete octahedral field $\mathcal{O}$ that is boundary-aligned and matches the singularity graph $\mathcal{S}$.

Edges and dual edges of $\mathcal{T}$ are equipped with an arbitrary but fixed orientation. Given the end hex-meshing goal, we assume our input $\mathcal{S}$ satisfies the local necessary conditions from Chapter 2.1 and the global necessary condition (2.3). In particular, $\mathcal{S}$ assigns hexahedral singularity types $\mathcal{S}(e_i) \in \{-1/4, 0, 1/4\}$ to each (oriented) edge $e_i \in E$, and $\mathcal{S}(v_i) \in \mathcal{H}_V$ to each vertex $v_i \in V$. The singular node type $\mathcal{S}(v_i)$ defines the entire local topology of incident singular edges including ordered triplets of singular edges that locally form the corner of a hexahedron. An example of an input singularity graph including singular node topology is shown in Fig. 5.2.

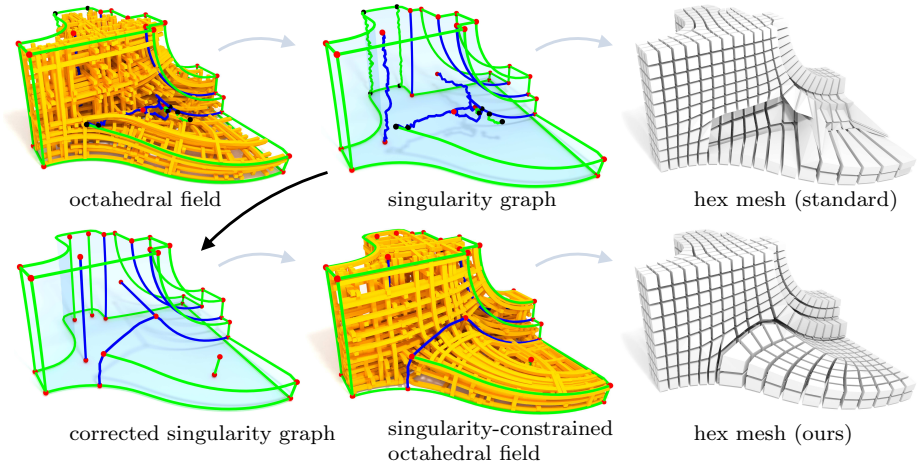
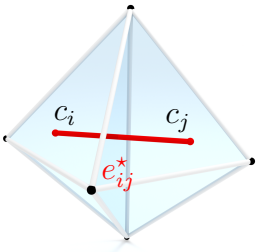


Figure 5.1: Overview. Octahedral fields often exhibit singularities that are invalid for hex meshing, inducing non-hex elements and poor quality (top row). Our algorithm (bottom row) starts from a corrected singularity graph and generates a meshable oct. field, resulting in a valid and less distorted hex mesh.

We assume the octahedral field to be induced by an (unknown) IGM, where each tetrahedron defines its own chart, as discussed in Section 4.3. Consequently, the octahedral field $\mathcal{O} = (\mathcal{R}, \mathcal{F})$ can be encoded as a set $\mathcal{R}$ of matchings $d\tau_{i\to j} = R_{i\to j} \in \mathrm{Oct}$ for (oriented) dual edges $e^\star_{ij} \in E^\star$ from cell i to cell j, and a set of frames $\mathcal{F}$ with $F_i \in \mathbb{R}^{3\times 3}$ belonging to cell $c_i \in C$.



Splitting the Input Mesh We require that none of the tetrahedra in $\mathcal{T}$ is adjacent to multiple singular edges or multiple boundary faces. Furthermore, we require that interior singular edges cannot be incident to tetrahedra with a boundary face, that a singular edge cannot be incident to two nodes of the singularity graph, and that a regular edge cannot be adjacent to two singular edges at both vertices. All these requirements can be easily satisfied by a series of edge splits in $\mathcal{T}$. A key property of the resulting mesh is that each singular edge has an independent fan of surrounding incident tetrahedra.

5.1.1 Singularity Graph Constraints

The problem of finding an octahedral field $\mathcal{O}$ that exhibits a given singularity graph $\mathcal{S}$ can be formulated as a (nonlinear) algebraic system of constraints for the unknown frames $\mathcal{F}$ and matchings $\mathcal{R}$. The first set of constraints is an immediate consequence of IGMs and ensures alignment of the octahedral field to boundary normals and singular arcs:

(C1) Boundary alignment At the boundary, the octahedral field aligns to the surface normal, i.e. for each boundary tetrahedron c_i with surface normal

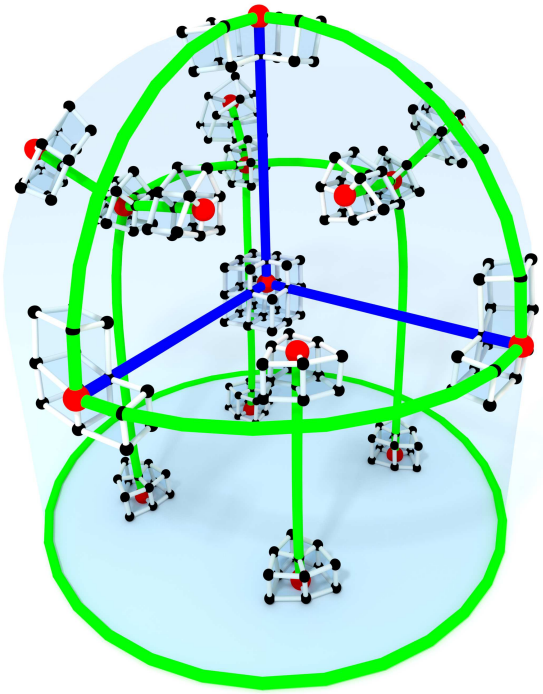


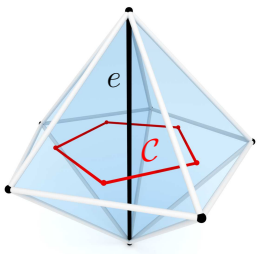
Figure 5.2: Visualization of a singularity graph $\mathcal{S}$, used as input for our algorithm. Green and blue curves are singular arcs of index $\frac{1}{4}$ and $-\frac{1}{4}$ respectively. Red spheres are singular nodes and their topological type $\mathcal{S}(v_i) \in \mathcal{H}_V$ is visualized as the corresponding hex-mesh. Note that except for singular edges, $\mathcal{S}$ does not constrain the geometric embedding at nodes.

n_i and frame F_i we require $n_i \in \mathcal{A}(F_i)$.

(C2) Singular arc alignment At singular arcs, the octahedral field is tangential, i.e. for each tetrahedron t_i adjacent to a singular edge e_j we require $e_j \in \mathcal{A}(F_i)$.

The second set of constraints, which incorporate matchings $\mathcal{R}$, ensures that for each point in space the octahedral field topology agrees with the topology of the prescribed singularity graph. The topology of the discrete octahedral field is measured in the one-ring tetrahedra incident to an edge or a vertex.

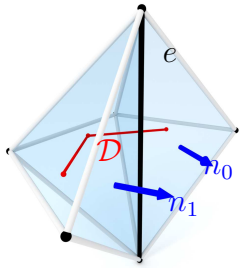
(C3) Edge type The hexahedral edge type $\mathcal{S}(e)$ specifies the *holonomy* of each dual parametric cycle $\mathcal{C}$, which circles the edge e counter-clockwise in its one-ring neighborhood. Assuming that the dual cycle $\mathcal{C}$ starts and ends in tet t with frame F and traverses matchings of dual edges in the order $R_0 \to R_1 \to \ldots \to R_k$, the holonomy is given as the product of these matchings, leading to the condition



$$R_k \ldots R_1 R_0 = \mathrm{rot}(F^{-1}\mathbf{e}, 2\pi\mathcal{S}(e)) \qquad (5.1)$$

where $\mathrm{rot}(\mathbf{a}, \alpha)$ is a rotation around axis $\mathbf{a}$ by angle α. This condition depends on the frame F, which is inevitable since we need to know the parametric coordinate axis to which $\mathbf{e}$ aligns to specify the correct holonomy. More precisely, while the angle $2\pi\mathcal{S}(e)$ is independent of the specific IGM, the rotation axis $F^{-1}\mathbf{e}$ is not, since the IGM could map e to 6 different coordinate axes, each inducing a different constraint.

For edges at the boundary we similarly measure the rotation of the surface normals in the parametric domain by means of an open dual path $\mathcal{D}$ traversing matchings $R_0 \to R_1 \to \ldots \to R_k$ between start frame F_0 and end frame F_1. Since the dual path is open, the surface normals of start and end tetrahedra might correspond to different axes of the coordinate frame, leading to a slightly modified equation



$$R_k \ldots R_1 R_0 F_0^{-1} [\mathbf{n}_0, \mathbf{e}] = \mathrm{rot}(F_1^{-1}\mathbf{e}, 2\pi\mathcal{S}(e)) \cdot F_1^{-1} [\mathbf{n}_1, \mathbf{e}] \tag{5.2}$$

Eq. (5.2) is valid for both interior and boundary edges of any index, since for interior edges $F_0 = F_1$ and $\mathbf{n}_0 = \mathbf{n}_1$ is an arbitrary vector orthogonal to $\mathbf{e}$. Regular edges with $\mathcal{S}(e) = 0$ do not align to coordinate axes but are nevertheless handled correctly since $\mathrm{rot}(\mathbf{a}, 0) = I_{3\times3}$, independently of $\mathbf{a}$.

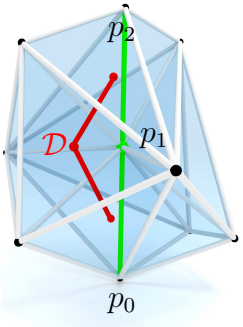
(C4) Vertex type While in the one-ring neighborhood of edges there is only a single rotation degree of freedom, the situation gets more complicated at vertices. The singularity graph defines constraints on the angles in which individual singular edges meet at vertices as well as their precise spatial orientation (e.g. three singular edges form a right-handed corner in frame space). For vertices of the singularity graph an additional set of constraints is required to ensure that all pairwise relations between adjacent singular edges agree with the singularity graph topology. The set of such singular vertex constraints can be decomposed into three different subtypes, the first relating tuples of collinear singular edges, which we call *tangent continuity constraints*, the second relating triples that form a corner in frame space, referred to as *corner constraints*, and the third called *sector constraints* which are the analogue of corners but on the boundary surface.

(C4a) Tangent continuity constraint Along an interior singular arc a consistent axis of the octahedral field is tangential, i.e. no corners or turning points are allowed, where the parametric alignment axis would change. Given a dual path $\mathcal{D}$ connecting two singular edges $\mathbf{e}_0 = \mathbf{p}_1 - \mathbf{p}_0$ and $\mathbf{e}_1 = \mathbf{p}_2 - \mathbf{p}_1$ which are adjacent at a common vertex p_1, tangent continuity means that

$$R_{\mathcal{D}} F_0^{-1}\mathbf{e}_0 = F_1^{-1}\mathbf{e}_1 \tag{5.3}$$

where $R_{\mathcal{D}}$ is the product of matchings along the dual path $\mathcal{D}$ as before and F_0, F_1 are the frames at start and end of $\mathcal{D}$.

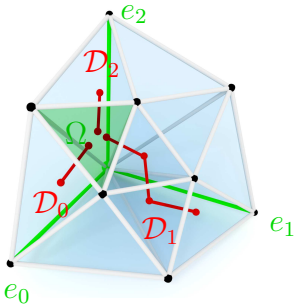
The idea is to express the frame axes corresponding to $\mathbf{e}_0$ and $\mathbf{e}_1$ both in the chart of F_1, where they can be compared. Eq. (5.3) must be satisfied for all dual paths $\mathcal{D}$ that do not enclose other singular arcs.



(C4b) Corner constraint The neighborhood of a singular node can be decomposed into (ordered) triplets of singular edges that form the right-handed corner of a hexahedron, e.g. four such corners for the $(4,0,0)$ node type. Each corner corresponds to a triangle in the sphere representation of Fig. 2.4.

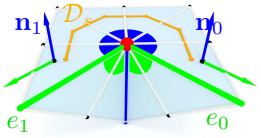
Expressing the parametric representation of all three outgoing singular edges frame in a common chart, results in a right-handed inducing the constraint

$$\left[R_{\mathcal{D}_0}F_0^{-1}\mathbf{e}_0 | R_{\mathcal{D}_1}F_1^{-1}\mathbf{e}_1 | R_{\mathcal{D}_2}F_2^{-1}\mathbf{e}_2\right] \in \text{Oct} \qquad (5.4)$$



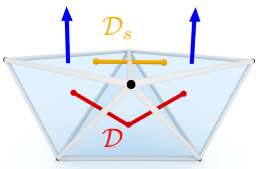
where edge e_i adjacent to frame F_i is transformed along (red) curve $\mathcal{D}_i$ to the (green) common chart Ω for comparison. The constraint should be satisfied for all combinations of curves $\mathcal{D}_i$ inside the region of the corner and not surrounding any singularity. It is enough to satisfy the constraint for a specific set of curves in the region in addition to having zero holonomy on regular edges.

(C4c) Boundary sector constraint At a boundary singular node, the one-ring neighborhood of surface triangles is partitioned into sectors formed by the incident singular boundary edges. A sector can be convex, flat, concave or a turning point, corresponding to interior angles of $k \cdot \frac{\pi}{2}$ with $k \in 1,2,3,4$ in parametric space respectively. Assuming that a sector is spanned counter-clockwise w.r.t. the surface normal from edge $\mathbf{e}_0$ to edge $\mathbf{e}_1$, which are connected by a dual surface path $\mathcal{D}_s$, a sector constraint is expressed through



$$R_{\mathcal{D}_s}F_0^{-1}[\mathbf{e}_0,\mathbf{n}_0] = \text{rot}\left(F_1^{-1}\mathbf{n}_1, k\cdot\frac{\pi}{2}\right)F_1^{-1}[\mathbf{e}_1,\mathbf{n}_1]. \qquad (5.5)$$

The upper inset figure shows three boundary sectors, adjacent to the central red vertex. Two sectors have an interior angle of $\frac{\pi}{2}$ (green) and one has an interior angle of $\frac{3\pi}{2}$ (blue). Eq. (5.5) is applied along the yellow path $\mathcal{D}_s$ to match the pairs of normal and edge vectors with the $\frac{3\pi}{2}$ sector angle. This uniquely specifies the rotation $R_{\mathcal{D}}$. In terms of matchings, each dual surface edge of $\mathcal{D}_s$ represents a chain of interior dual path $\mathcal{D}$ as shown in the second inset, i.e. $R_{\mathcal{D}_s} = R_{\mathcal{D}}$.



In summary, octahedral node topology at a tet mesh vertex is fully defined by an overlapping set of corner, sector, and tangent continuity constraints that are induced by the corresponding hex mesh vertex type.

Algebraic system A singularity graph $\mathcal{S}$ induces the following set of constraints: For each boundary triangle of $\mathcal{T}$ one (C1) constraint, for each edge of $\mathcal{T}$ one constraint of type (C3), if singular additionally one of type (C2), and a set of node constraints (C4) for each vertex of $\mathcal{T}$ which is adjacent to at least one singular edge. The number of (C4) constraints at a vertex depends on the node type described in Chapter 2.1. While some of these constraints may be

globally redundant with each other, they are necessary and sufficient for correct local topology.

Finding solutions to the algebraic system is difficult since the number of constraints is large ($\geq 2 \cdot |E|$), the constraints are nonlinear (products in the group of rotations), and the problem involves continuous as well as discrete variables (frames are continuous, matchings and choice of alignment axes are discrete). In the following, we describe a novel algorithm for this task, which based on a careful analysis leverages several structural properties of the algebraic system.

5.1.2 Decomposition approach

One difficulty of the algebraic system results from the diversity of involved variables (frames and matchings). However, the problem is highly underdetermined, providing flexibility to *a priori* fix alignment of singular edges where required for the solution, leading to a reduced, purely discrete algebraic system only involving the matchings. Once all matchings are determined, the complete set of frames can be found in an independent subsequent step described in Section 5.1.4. Our decomposition approach is based on the following observation:

Observation 1 There are $24^{|C|}$ topologically identical representations of a discrete octahedral field with different matchings and frames at faces and cells of a tetrahedral mesh $\mathcal{T}$. This becomes obvious when considering that there are 24 different frames representing one element of an octahedral field and that inversely transforming a frame and its matchings to neighbors does not alter field topology. Such a topology preserving transformation is simply a change of a local coordinate system.

As an immediate consequence, for every octahedral field there is a choice of coordinate systems, where each singular edge $\mathbf{e}_i$ aligns in all charts of its incident tets t_j to the u-direction, and at the same time each boundary normal $\mathbf{n}_j$ aligns in the chart of its boundary tetrahedron to the v-direction, i.e. the parametric images of singular edges and normal vectors simplify to $F_j^{-1}(\mathbf{e}_i) = \hat{\mathbf{u}}$ and $F_j^{-1}(\mathbf{n}_j) = \hat{\mathbf{v}}$. This is possible since as mentioned before, our tetrahedral mesh is always split such that no tetrahedron is incident to more than one singular edge, and no boundary tetrahedron is incident to an interior singular edge. Our specific choice of coordinate systems greatly simplifies the algebraic system. Conditions (C1) and (C2) are satisfied by construction, and for (C3) all edge holonomies $H_e = \mathrm{rot}(\hat{\mathbf{u}}, 2\pi\mathcal{S}(e))$ are fully determined, simplifying Eq. (5.2) to

$$R_k \dots R_1 R_0 = H_e \tag{5.6}$$

with known righthand side and independent of the frames $\mathcal{F}$. In the same way, Eqs. (5.3), (5.4) and (5.5) are simplified to

$$R_{\mathcal{D}}\hat{\mathbf{u}} = \pm\hat{\mathbf{u}}, \tag{5.7}$$

$$[R_{\mathcal{D}_0}\hat{\mathbf{u}}|R_{\mathcal{D}_1}\hat{\mathbf{u}}|R_{\mathcal{D}_2}\hat{\mathbf{u}}] \in \mathrm{Oct}, \tag{5.8}$$

and

$$R_{\mathcal{D}} = \mathrm{rot}\left(\hat{\mathbf{v}}, \pm k \cdot \left(-\frac{\pi}{2}\right)\right), \tag{5.9}$$

where we exploit the additional convention that the canonical orientation of singular edges is always outward-pointing from nodes of the singularity graph. Choice of the sign in Eqs. (5.7) and (5.9) is fully determined by the canonical orientation of singular edges: it is positive if at the common vertex one edge is incoming and the other outgoing, otherwise negative.

In summary, the advantage of our specific choice of coordinate systems is the simplified algebraic system consisting of Eqs. (5.6), (5.7), (5.8), and (5.9), which solely depends on the matchings $\mathcal{R}$.

5.1.3 Determining matchings

To find a set of matchings that satisfy the simplified algebraic system, we design a **chart-merging** algorithm. It is based on two principles. (1) In the beginning all matchings are unknown which means that each tetrahedron forms a separate chart. (2) Whenever a matching of a dual edge is determined, we interpret this as merging the charts of both neighboring tetrahedra. The mathematical challenge lies in *locally* determining matchings that are *globally* consistent.

Constrained chart-merging A frequent operation in our algorithm is to merge two charts while satisfying a constraint on how specific coordinate axes $\mathbf{a} \in \{\pm\hat{\mathbf{u}}, \pm\hat{\mathbf{v}}, \pm\hat{\mathbf{w}}\}$ match along a dual path. The operation is illustrated in Fig. 5.4. More precisely, assume two separate charts A and B are connected by a dual path $\mathcal{D}$ with combined matching $R_{\mathcal{D}} = R_k \dots R_0$. Some of these matchings might already be fixed but since the charts are separate there is at least one R_j that is not yet determined. First, all undetermined matchings on $\mathcal{D}$ other than R_j are set to identity. Then the constrained chart-merging is performed by choosing R_j such that the constraint is satisfied. If the matching of two coordinate axes is specified the combined matching is unique $R_{\mathcal{D}} = R$ and R_j is obtained through

$$R_j = (R_k \dots R_{j+1})^{-1} R (R_{j-1} \dots R_0)^{-1} \tag{5.10}$$

If matching of only one coordinate axis is specified, i.e. $R_{\mathcal{D}}\mathbf{a} = \mathbf{b}$, we obtain R_j from

$$R_j[(R_{j-1} \dots R_0)\mathbf{a}] = (R_k \dots R_{j+1})^{-1}\mathbf{b} \tag{5.11}$$

We arbitrarily choose one of the four $R_j \in \mathrm{Oct}$ which maps $(R_{j-1} \dots R_0)\mathbf{a}$ to $(R_k \dots R_{j+1})^{-1}\mathbf{b}$. Finally, if no matching of coordinate axes is constrained, each $R_j \in \mathrm{Oct}$ is a valid solution and we always pick the identity.

Chart-zippering The second central operation besides chart-merging is chart-zippering. Given an edge e with all but one matching of incident triangles known, the unknown matching R_j can be uniquely determined based on the edge holonomy Eq. (5.6):

$$R_j = (R_k \dots R_{j+1})^{-1} H_e (R_{j-1} \dots R_0)^{-1} \tag{5.12}$$

The operation is called zippering in analogy to the zippering algorithm of [RVLL08] for N-symmetry fields on surfaces with prescribed singularities. In our terms, the algorithm of [RVLL08] creates one initial connected chart through

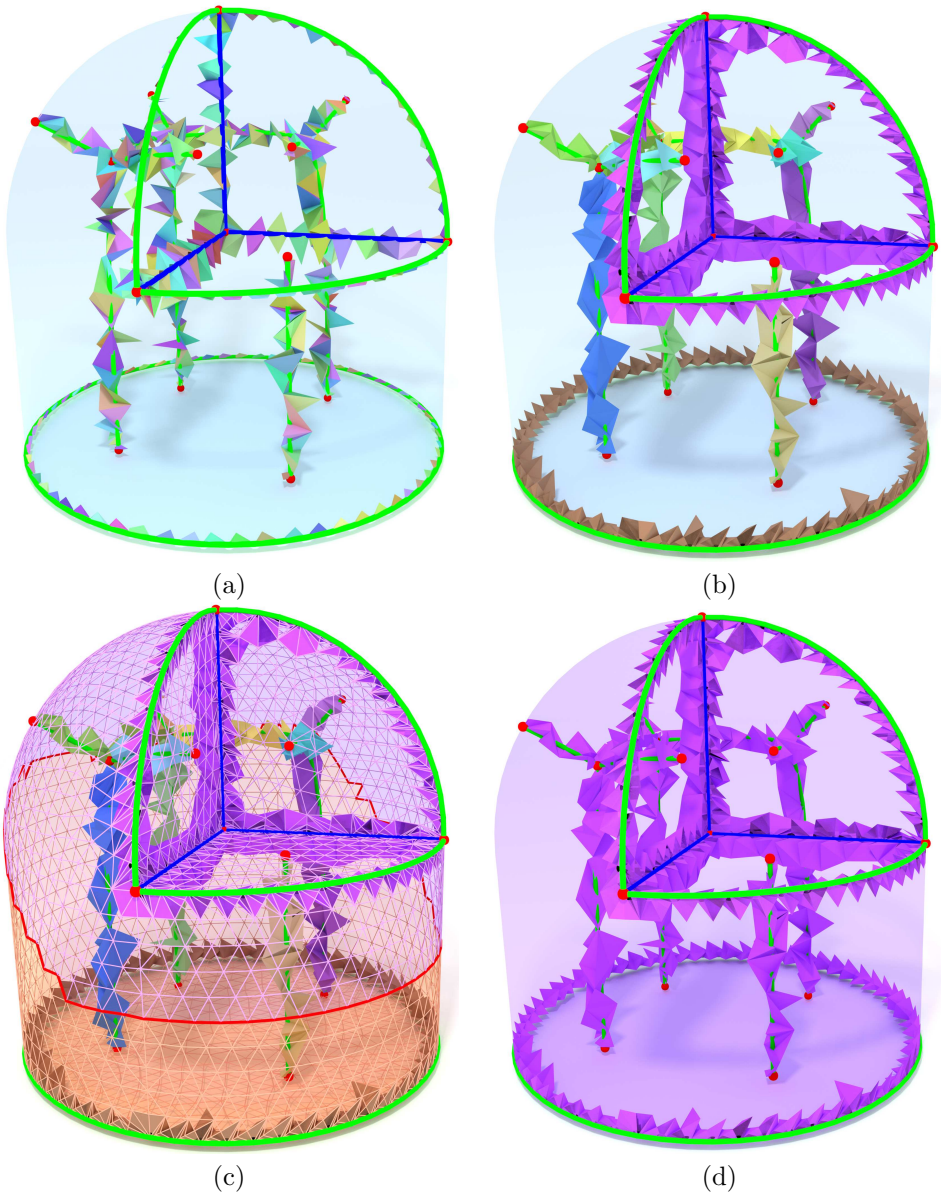


Figure 5.3: Algorithm Overview: (a) A separate initial region is created for each singular edge. (b) Regions are merged into tubes corresponding to singular arcs and tube networks belonging to boundary components of the singularity graph. (c) Interior tubes touching the boundary are connected to the boundary components by chart-zippering on the boundary shell. (d) the algorithm terminates after all charts are merged and all matchings are determined.

a dual spanning tree of surface triangles and then iteratively performs chart-zippering until all matchings are determined. In contrast to the simpler surface case we need to cope with (C4) node constraints which render a simple spanning tree construction infeasible and require a carefully designed chart-merging.

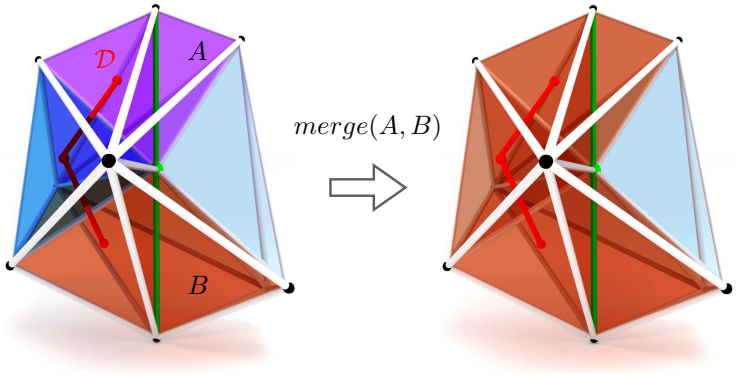


Figure 5.4: Chart merging: two individual charts A and B are merged through a dual path $\mathcal{D}$.

Algorithm The algorithm performs chart-merging in three major steps outlined in Algorithm 1, to determine the set of matchings X. The idea is to proceed from singular arcs and boundary surfaces, where most information in the form of constraints is available, to the interior of the volume. First charts incident to singular edges are merged along singular arcs, leading to a set of *singular tubes* illustrated in Fig. 5.3b. In the second step the charts of boundary tets are merged, Fig. 5.3c. Finally, the algorithm extends the information to the interior until all matchings are determined, Fig. 5.3d.

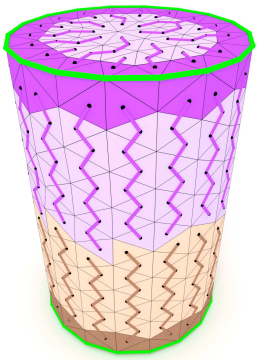
Algorithm 1 DetermineMatchings

Input: Tet mesh with singular topology constraints (5.6)-(5.9)
Output: matchings $X \in \text{Oct}^{|T|}$ satisfying (5.6)-(5.9) or INFEASIBLE

1: $X \leftarrow$ flag all matchings as uninitialized
2: $X \leftarrow$ MergeSingularArcCharts(X) ▷ Singular tube charts
3: $X \leftarrow$ MergeBoundaryCharts(X) ▷ Boundary shell charts
4: **return** MergeVolumeChart(X) ▷ Solve remaining matchings

Step 1: Merging charts of singular arcs We begin with generating one combined chart for each singular edge. For one singular edge, the set of incident matchings $R_0 \dots R_k$ needs to satisfy the holonomy constraint Eq. (5.6). Since singular edges are consistently oriented in all incident charts, it is sufficient to set $R_1 \dots R_k$ to identity and determine R_0 through chart-zippering Eq. (5.12). An example of the resulting singular edge charts is depicted in Fig. 5.3a. Next we connect charts along singular arcs. For interior singular arcs, each pair of neighboring singular edges is equipped with a tangent continuity constraint of Eq. (5.7), allowing a series of constrained chart-merging operations to obtain one tubular chart per interior singular arc. Similarly, we perform constrained chart-merging for all boundary sector constraints (5.9), leading to a closed tubular network for each connected component of the boundary singularities. Pseudocode of these steps is provided in Algorithm 2 and an example visualization of resulting charts is depicted in Fig. 5.3b.

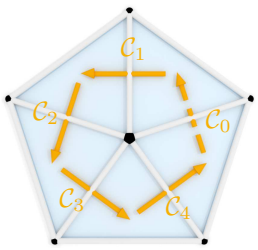
Step 2: Merging charts at boundary surfaces In this step we reduce the number of independent charts on the boundary. The key observation is that away from singular edges a consistent frame axis is aligned to the surface normal, reducing the octahedral field topology to the 2D cross-field case. Consequently, we can adapt the 2D zippering approach of [RVLL08] to consistently merge boundary charts. We process each connected boundary surface independently. First, the boundary triangles of charts containing singularities are used as the root set of a dual spanning forest of all boundary triangles, as shown on the right. For boundary surfaces without singular edges, we choose one arbitrary triangle as the root. Next, for each dual boundary edge d of the spanning tree we merge the charts of both incident triangles by constrained chart-merging for the corresponding interior dual path $\mathcal{D}$.



Algorithm 2 MergeSingularArcCharts(X)

```
for all singular edges e do
    X ← MergeAndZipChart(e)                    ▷ Singular edge charts
end for
for all tangent continuity constraints T do
    if T relates two different charts then     ▷ Prevent cycles
        X ← ConstrainedChartMerging(T)         ▷ Int. sing. tubes
    end if
end for
for all boundary sector constraints B do
    X ← ConstrainedChartMerging(B)             ▷ Bnd. sing. tubes
end for
return X                                       ▷ Return determined matchings
```

Since spanning forest edges are never singular, both normals agree in a common chart and we constrain $R_{\mathcal{D}} = I_{3\times 3}$. In the final step, we iteratively apply 2D chart-zippering, where the boundary vertex index needs to be taken into account when closing dual cycles. Assume a counter-clockwise (ccw) cycle of dual (non-singular) boundary edges $d_0 \dots d_k$, which surround vertex p and where the matchings on $\mathcal{D}_i$ are known for all i except one. The holonomy of the cycle must be a rotation around the coordinate axis of the normal vector with angle given by the index of the boundary vertex, i.e. $R_{\mathcal{D}_k \dots \mathcal{D}_0} = \mathrm{rot}(\hat{\mathbf{v}}, 2\pi \cdot \mathrm{idx}(p))$, leading to one step of constrained chart-merging. The ccw index $\mathrm{idx}(p)$ of a boundary vertex p is equal to the index of an incident interior singular edge or zero if there is none. In each boundary component without singular edges all boundary tetrahedra will be merged into a single chart. If a boundary surface has k connected sets of singular boundary edges, 2D chart-zippering will terminate with k independent charts, which will only be merged in a subsequent step. Pseudocode of the boundary chart merg-

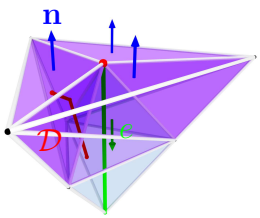


ing is provided in Algorithm 3 and Fig. 5.3c shows an example resulting in two boundary charts that are separated by the red boundary curve.

Algorithm 3 MergeBoundaryCharts(X)

```
for all boundary components B do
    Construct dual spanning forest DSF on boundary
    for each dual edge d ∈ DSF do
        X ← ConstrainedChartMerging(D, I_{3×3})
    end for
    while ∃ zippable ccw-cycle C enclosing vertex p do
        X ← ConstrainedChartMerging(C, rot(v̂, 2π · idx(p)))
    end while
end for
return X                                   ▷ Return determined matchings
```

Step 3: Merging charts in the volume In this step we merge interior singular tubes to the boundary charts, including those indirectly linked through corners of the singularity graph. First, for each boundary surface chart we merge all interior singular tubes that already touch the boundary through a vertex. This can be easily done with a constrained chart-merging that matches the $\hat{\mathbf{v}}$ coordinate axis of the outward-pointing surface normal $\mathbf{n}$ of the boundary chart with the negative of the inward-pointing coordinate axis $\hat{\mathbf{u}}$ of the singular edge e, i.e. with the matching $\mathrm{rot}(\hat{\mathbf{w}}, -\pi/2)$. Singular tubes that touch the boundary at both ends are only merged on one side, since we cannot decide the twist of global cycles at this point.



Next, we grow a dual spanning forest rooted at boundary chart tetrahedra to all tetrahedra not part of a boundary or tube chart yet. Unconstrained chart-merging is performed for all dual edges of the spanning forest, i.e. matchings are set to identity. Additionally, iterative chart-zippering is used to resolve all locally decidable matchings. After these two steps, all remaining charts are either boundary charts or isolated singular tube charts that do not touch the boundary.

Further singular tubes can be merged to the boundary charts by considering corner constraints (5.8). A corner constraint is formed by three dual paths $\mathcal{D}_0$, $\mathcal{D}_1$ and $\mathcal{D}_2$ that express the axis orientation of the three singular edges e_0, e_1 and e_2 of a corner in a common chart. We say that a corner constraint is *decidable*[1] if (i) two singular edges are part of a common boundary chart and all matchings on their $\mathcal{D}_i$ are determined and (ii) the third singular edge is not part of a boundary chart. Assuming w.l.o.g. that e_2 is the non-boundary chart edge, we connect it to the boundary by constrained chart-merging along $\mathcal{D}_2$ with matching constraint

$$R_{\mathcal{D}_2}\hat{\mathbf{u}} = R_{\mathcal{D}_0}\hat{\mathbf{u}} \times R_{\mathcal{D}_1}\hat{\mathbf{u}} \tag{5.13}$$

[1] Another special case of decidable corner arises, if one singular edge belongs to a boundary chart not containing boundary singular edges and the other two belong to isolated int. singular tubes. Any locally valid matchings for $\mathcal{D}_i$ are then globally valid.

Algorithm 4 MergeVolumeChart(X)

for all interior singular tubes T **do**
  if T incident to boundary **then**
    $X \leftarrow$ ConstrainedChartMerging(T) ▷ Connect to bnd.
  end if
end for
Grow dual spanning forest DSF from boundary chart tetrahedra
for all dual edges $d \in DSF$ **do**
  $X \leftarrow$ UnconstrainedChartMerging(d)
end for
$X \leftarrow$ Chart-Zippering(X) ▷ Solve locally decidable match.
$B \leftarrow \varnothing$ ▷ Set of unfinished branches
loop
  if $\exists$ decidable corner C **then**
    $X \leftarrow$ ConstrainedChartMerging(C) ▷ Eq. (5.13)
  else
    if $\exists$ decidable patch P **then**
      $X \leftarrow$ ConstrainedChartMerging(P)
    else
      $(i, \{m_1 \dots m_k\}) \leftarrow$ FindBestPatch(X) ▷ $k \in \{4, 24\}$
      for $j = 2 \dots k$ **do**
        $X_i \leftarrow m_j$
        $B \leftarrow B \cup \{X\}$ ▷ Store candidate for later
      end for
      $X_i \leftarrow m_1$ ▷ Proceed with first candidate
    end if
  end if
  $X \leftarrow$ Chart-Zippering(X) ▷ Solve locally decidable match.
  if #charts= 1 **then**
    $X \leftarrow$ HandleUnsolvedMatchingsByBranching(X)
    if X satisfies all constraints (5.6)-(5.9) **then**
      return X ▷ Valid solution found
    else
      if $B \neq \varnothing$ **then**
        $X \leftarrow$ get next branch of B ▷ Continue search
      else
        return INFEASIBLE
      end if
    end if
  end if
end loop

directly following from (5.8). After merging a corner, we perform iterative chart-zippering, which frequently has the positive effect of resolving matchings that render other singular corners decidable. Often, iteratively performing the combination of merging at a locally decidable corner with subsequent chart-zippering is sufficient to resolve all matchings.

If no decidable corner constraint exists, non-local considerations are required to proceed. To bundle spatially distributed constraints, we first partition the triangles of unsolved matchings $T_U \subset T$ into subsets that represent identical degrees of freedom. These are exactly the 2-manifold patches embedded in T_U since fixing the matching of one triangle in such a patch is sufficient to resolve all others by chart-zippering. A patch is *decidable* if it is supported by two independent matching constraints, which uniquely specify the single matching degree of freedom and enable constrained chart-merging.

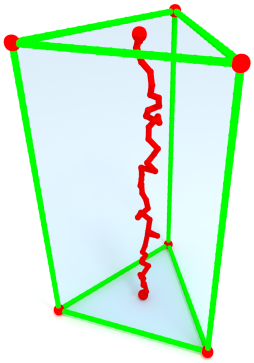
If neither decidable corners nor patches are available, we explore all potential solutions until a valid one is found. The idea is to search a maximally constrained patch with minimal potential solutions, which are either 4 if it is supported by a matching constraint or 24 otherwise. We randomly pick one candidate solution and continue the algorithm. All other candidates are stored as unfinished branches for later investigation. If the current candidate turns out to be infeasible, we backtrack and search other branches. The same branching strategy is used if the final chart-zippering cannot resolve all matchings. This exhaustive search ensures that whenever a solution exists, it is found. Complete pseudocode of the volumetric chart merging is provided in Algorithm 4.

Algorithm Properties The input to the algorithm is a locally meshable singularity graph that also satisfies the global necessary condition. The output is a set of matchings of a locally meshable octahedral field if it exists, otherwise the message "INFEASIBLE" is returned. The driving principle of the algorithm is to in each step identify a constraint that can be used to resolve a matching while maintaining feasibility regarding all other constraints. In this sense, it shares similarity with triangular solvers for linear systems of equations, which analogously in each step solve for one variable or choose underdetermined degrees of freedom. Since we cannot guarantee that in each step we can solve for a unique matching, the algorithm additionally includes fallback to exhaustive search of a set of candidate matchings. In this way, termination with a valid solution is guaranteed if one exists. While theoretically necessary, in none of our experiments were multiple branches explored, leading to very fast runtimes in practice. Analyzing the underlying theoretical reason is an interesting direction for future work.

We observed that the singularity graph does not always uniquely specify the octahedral field topology. In particular for higher genus handlebodies, closed singular arcs inside the volume, or cavities, additional constraints are often necessary to uniquely specify field topology. This is not a problem for our algorithm, since by construction it is able to handle underdetermined cases. However, to obtain more control, one could either provide additional constraints or postpone the decision of additional degrees of freedom to the frame generation

to, e.g. exploit the degrees of freedom to obtain the smoothest solution.

Relation to Global Necessary Condition While the global necessary condition Eq. (2.3) is applied as a filter on the input singular graphs, and helps to detect global inconsistencies, it is not sufficient to guarantee existence of a compatible locally meshable field. Consider the inset example. While the input singularity graph (green lines) satisfies Eq. (2.3), our algorithm is able to detect that no corresponding locally meshable field exists. Our algorithm produces matchings with extra singularities (red lines) that do not agree with the input singularity graph. While this hints at a method for correcting the singular graph, it is outside the scope of this paper to correct singularity graphs.



5.1.4 Octahedral Fields with Fixed Matchings and Alignment

After determining the topological matchings and partial alignment information, we still need to construct the frames geometrically to obtain the octahedral field. Regardless of the geometric frame field output, the topology of the frame field is already specified by the topological matchings. However, for hexahedral meshing applications, we aim for the smoothest frame field with prescribed topology. Accordingly, we minimize the Dirichlet energy of a quaternion field q subject to alignment constraints for $q \in B$ resulting from normals and singular edge alignment

$$\begin{aligned} \underset{q}{\text{minimize}} \quad & \int_\Omega |\nabla q|^2 dV \\ \text{subject to} \quad & A_i q_i = 0,\ q_i \in B \\ & |q_i| = 1,\ i = 1 \dots n \end{aligned}$$

with $A_i \in \mathbb{R}^{2\times 4}$ are linear alignment conditions derived below. The $|q_i| = 1$ constraint may be ill-posed in the continuum, an issue addressed using the relaxation below.

Relaxation to eigenvalue problem The Dirichlet energy is expressed w.r.t. the connection specified by the matchings. Thus, if two orthogonal frames are related through $F_j = F_i R_{j\to i}$, the same relation can be expressed in quaternion form $q_j = q_i \hat{R}_{j\to i}$. While more sophisticated variants are possible, a uniformly weighted discretization of the Dirichlet energy summing squared differences for all dual edges $e^\star_{ij}$ is sufficient for our purposes:

$$\int_\Omega |\nabla q|^2 dV \approx \sum_{e^\star_{ij}} |q_i \hat{R}_{j\to i} - q_j|^2. \tag{5.14}$$

In the spirit of the globally optimal direction fields of [KCPS13] we relax the unit-norm constraints $|q_i| = 1$ to $\sum_i |q_i|^2 = n$, turning the optimality conditions of our optimization problem into an eigenvalue problem. We add penalty terms

$\omega_a |A_i q_i|^2$ to enforce the alignment conditions in the objective function; $\omega_a = 10$ already produces solutions that obey the constraints well.

Axis alignment conditions The alignment conditions are homogenous linear expressions resulting from the following observation. Assume that we want to align the u-axis of our frame represented by q_i to the direction v. In this case q_i is parametrized by $q_i = q_{\hat{\mathbf{u}} \to v} \cdot q_{\hat{\mathbf{u}}}(\alpha)$, where $q_{\hat{\mathbf{u}} \to v}$ is an arbitrary rotation that maps $\hat{\mathbf{u}}$ to v and

$$q_{\hat{\mathbf{u}}}(\alpha) = \left(\cos\frac{\alpha}{2}, \sin\frac{\alpha}{2}, 0, 0\right)^T \tag{5.15}$$

is a rotation around the first coordinate axis. Since quaternion multiplication is linear in each quaternion we can express the product as $q_i = Q\, q_{\hat{\mathbf{u}}}(\alpha)$, with $Q \in \mathbb{R}^{4\times 4}$ expressing multiplication from the left by $q_{\hat{\mathbf{u}} \to v}$. Hence for normal alignment we end up with the following four constraints on q_i:

$$Q^{-1}\, q_i = \left(\cos\frac{\alpha}{2}, \sin\frac{\alpha}{2}, 0, 0\right)^T$$

It suffices to impose the latter two homogenous constraints because any unit quaternion fulfilling the latter two conditions, i.e. of the form $(q^w, q^x, 0, 0)$, is automatically of the required form (5.15) for some α. The projection of a given quaternion q to the closest one satisfying a partial alignment condition q_a is done by

$$q_a = Q\,\mathrm{diag}(1, 1, 0, 0) Q^{-1} q$$

with subsequent normalization. We use projection to eliminate small constraint violations resulting from relaxation to penalty terms.

Coping with the double-covering Since quaternions double-cover SO(3), meaning that q and $-q$ represent the same rotation, there is one additional degree of freedom that we need to address. When specifying the transition functions $\pm\hat{R}_{j\to i}$ in quaternion form, the sign is undefined; hence, we need to ensure that we get the right connection on dual cycles in our mesh respecting the double-covering. To this end, we check and correct all dual edge cycles using an algorithm similar to the chart-zippering of Chapter 5.1.3. We first randomly initialize the signs of $\pm\hat{R}_{j\to i}$ and then whenever a dual cycle is closed, we check whether the first component of the quaternion product along the cycle is positive. If it is not, we invert the sign of the matching quaternion that closes the cycle. A negative sign in the first quaternion component indicates that the encoded holonomy identifies two different representations in the quaternion double cover. Luckily, the alignment conditions are homogenous and therefore simultaneously valid for $\pm q_i$; this makes them invulnerable to double-cover issues.

5.2 Results

We evaluate our algorithm by means of several example models of various complexity shown in Figs. 6.1, 2.1, 5.5 and 5.6. For all examples of Fig. 5.5 we obtained an initial singularity graph through an octahedral field generated with [RSL16]. Based on the necessary local and global conditions of Chapter 2.1 we

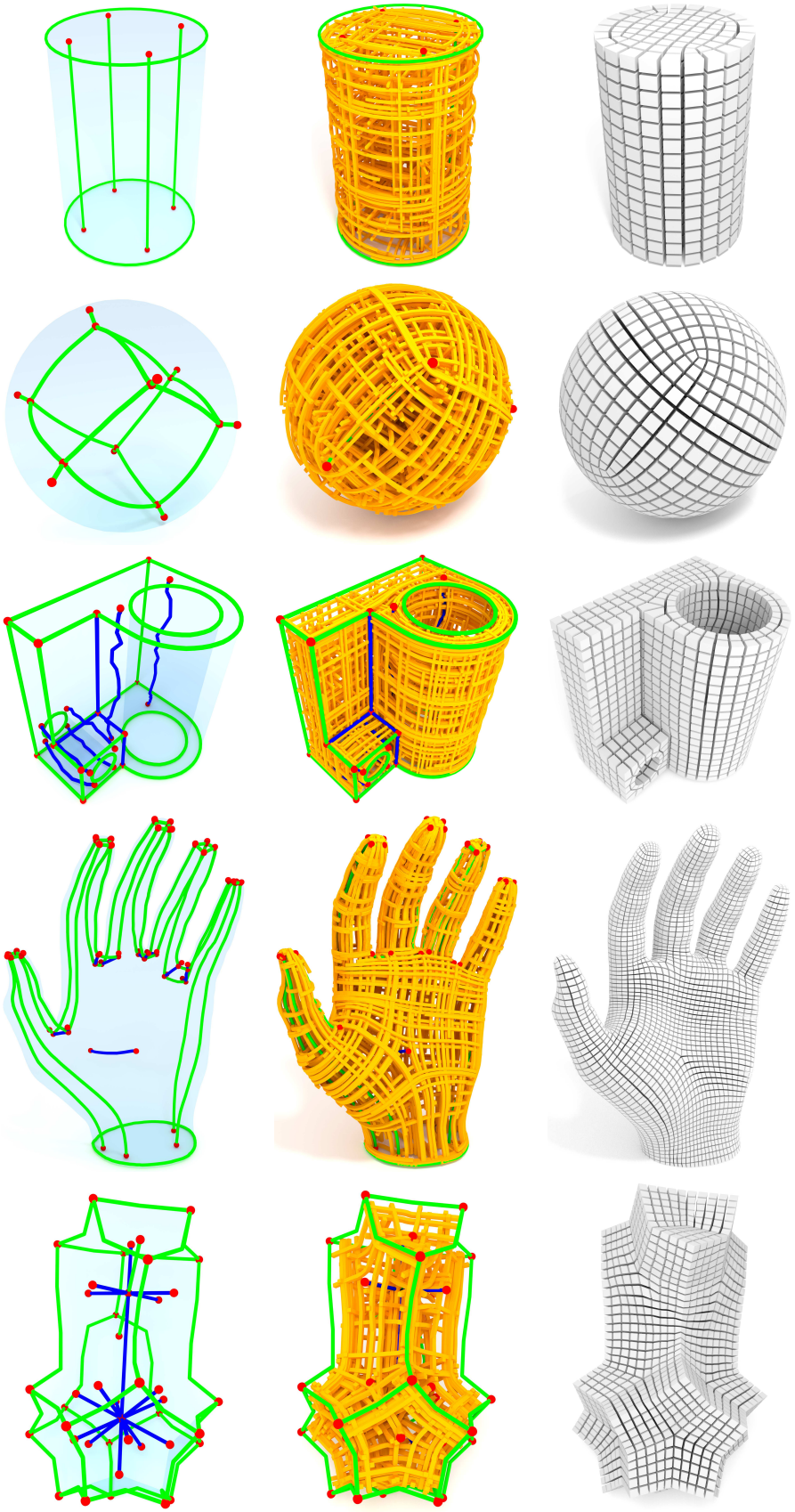
manually repaired the singularity graph and used the result as input for our algorithm. The resulting matchings and frames then serve as input for [NRP11] to obtain an integer-grid map, which is hex meshed with [LBK16].

The manual correction of a singularity graph was done by iteratively adding, removing and smoothing singular arcs until the singularity graph became locally meshable and satisfied the global necessary condition. If our algorithm reported that the singularity graph was still not globally meshable, we performed further modifications, inspired by the constraints that could not be satisfied. Algorithmic correction is out of the scope of this work but will be an important topic for the future. For the joint model in Fig. 5.5 we perturbed the input singular graph to demonstrate a result on messy input. Our algorithm only cares about the topology of the input singular graph such that geometrically messy input does not impact the extracted matchings.

The complex examples of Fig. 5.6 were generated in a similar manner, however, the singularity graphs were imported from hexahedral meshes provided in the supplemental material of [FXBH16, LLX$^+$12, FBL16, HJS$^+$14]. For all models that we tried, our algorithm generated a valid octahedral field with correct singularity graph as verified by checking the algebraic system for the output field; this empirically verifies correctness of our algorithm. The input complexity ranges from 5k up to 300k tetrahedra for the elephant model. Even for the largest models, generating the matchings only requires a few seconds, while the optimization problem to obtain the frames takes significantly longer, e.g. 40 seconds for the bunny model.

Comparison to previous work Since our method is the first one for generating octahedral fields with prescribed topology, a direct comparison to previous work is impossible. Closest to our method are the singularity correction techniques of [LLX$^+$12] and [JHW$^+$14]. While they are automatic, they only perform a small set of local corrections through splitting or collapsing singularities that is not sufficient to obtain meshable singularity graphs for most of our examples. For example, in Fig. 6.1, global changes and the addition of new singular arcs are necessary. Both previous methods cannot handle such global changes, which require a re-computation of the octahedral field.

The geometric quality of the hexahedral meshes, typically measured using scaled Jacobians, mostly depends on the pre-processing (design of singularity graph) and post-processing (optimization e.g. via [LSVT15]). Consequently, measuring scaled Jacobians is not a meaningful way to evaluate success of our algorithm, since we would mostly measure how much time we spent on tuning the input singularity graph. Our scaled Jacobians are typically on par with the state of the art.



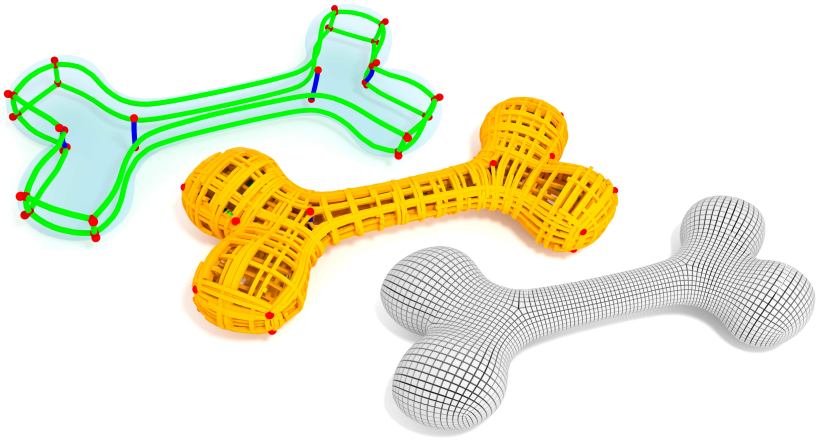
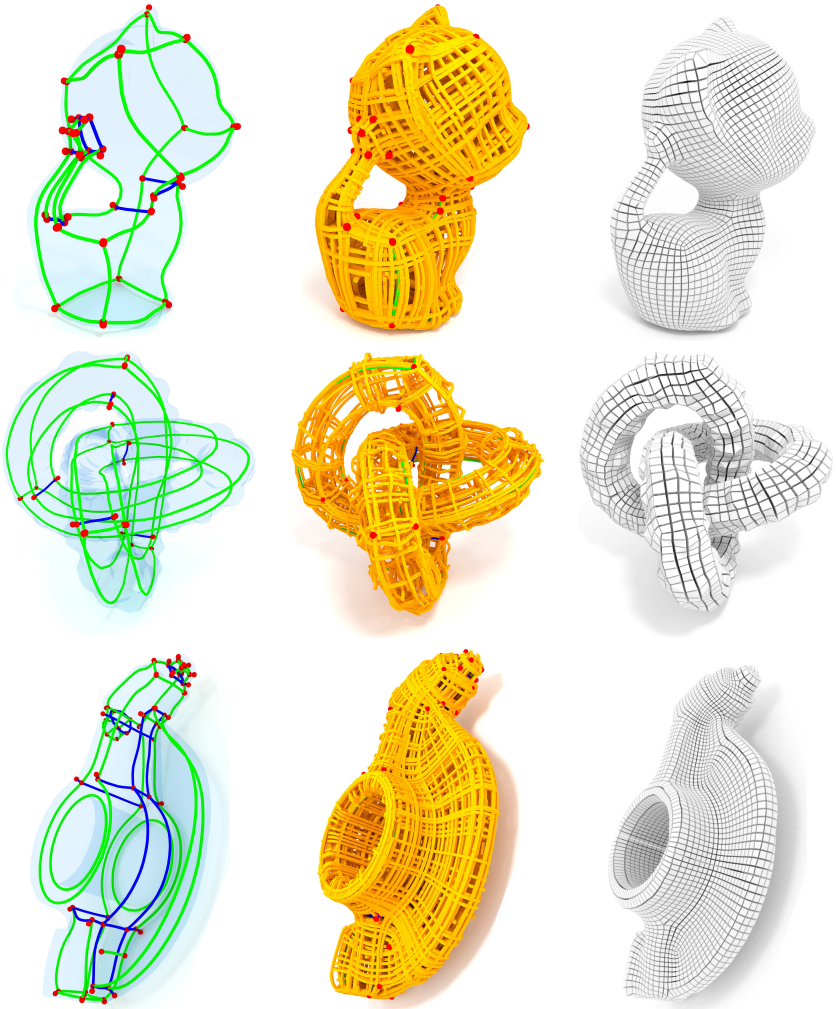


Figure 5.5: Several models with manually corrected singularity graph. From left to right: singularity graph, octahedral field and hexahedral mesh.



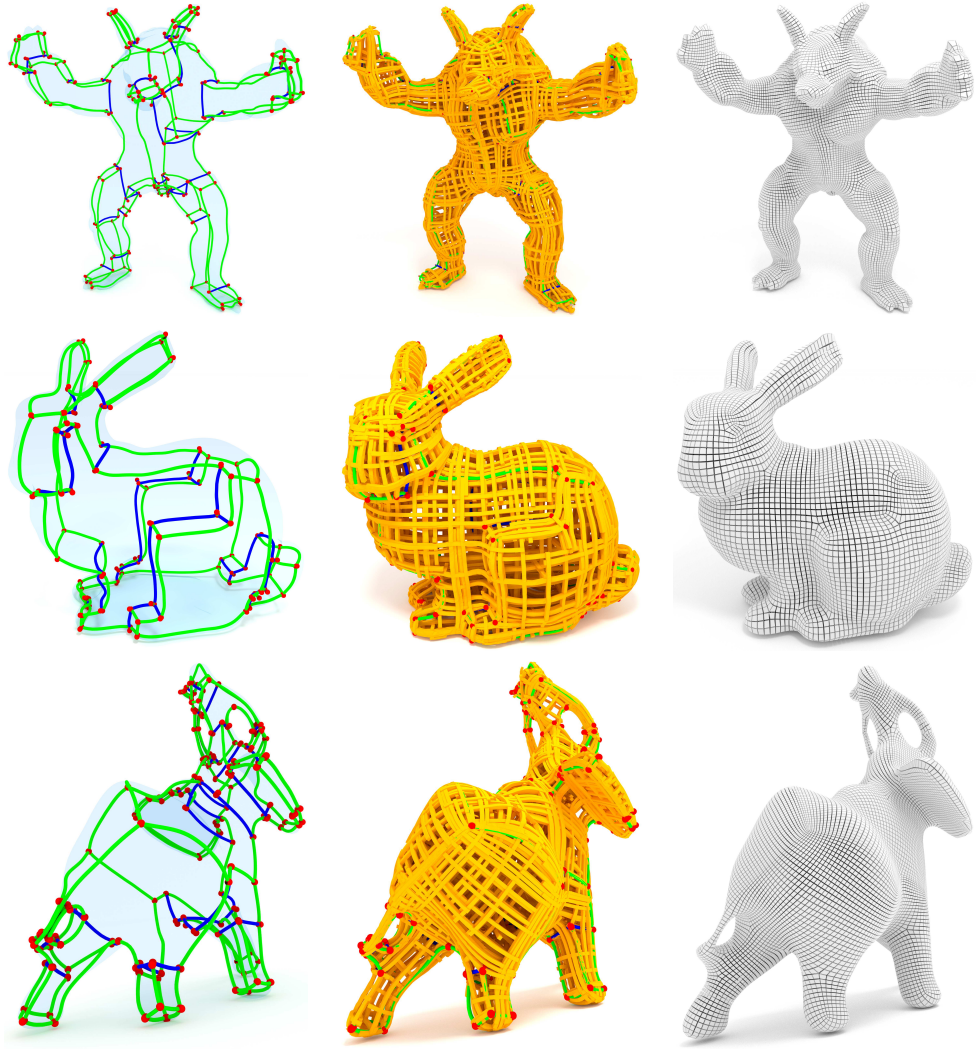


Figure 5.6: Complex examples obtained from supplemental material of: [FXBH16] kitten and knot, [LLX+12] rockerarm, [FBL16] bunny and armadillo, and [HJS+14] elephant.

5.3 Discussion

Many confounding factors make automatic hexahedral meshing extremely difficult; somehow, the elegant structure of two-dimensional quad meshing problems does not admit an obvious lifting to the volumetric case. While octahedral field-based meshing appears to be a strong contender for the design of practical and efficient algorithms, many questions remain in establishing the fundamentals of this approach. We consider our work to be a serious step toward theoretically-justified, robust field-based hex meshing. By returning to the basics, we clarify the fundamental unknowns in the problem by defining the relevant algebraic

system. Along the way we derive new necessary local and global conditions for hex meshability of singular graphs, including a *complete* enumeration of the practically-relevant vertex singularities. Furthermore, our *chart-merging* algorithm robustly recovers topology-constrained octahedral fields that can be provided to existing field-guided meshing techniques.

Several open mathematical and algorithmic challenges remain in the domain of field-guided hex meshing. The holy grail in this domain is a complete (necessary *and* sufficient) characterization of meshable singular octahedral field topologies, ideally accompanied by an algorithm that can project a singular graph to its closest meshable counterpart. A continuum theory of octahedral fields posed using differential geometry/topology language rather than relying on an underlying discrete structure may also provide insight. In parallel with these theoretical considerations, other topics for future research are more human-oriented. In particular, while singular topology is critical for understanding the set of hex meshes that can be embedded in a particular volume, it may be the case that a natural user interface for hex meshing should incorporate guidance in a different form, inferring the singular topology automatically behind the scenes. One intriguing direction might be to *learn* a map from volumes to singular graphs informed by a collection of hand-designed hex meshes.

These future improvements aside, we anticipate that our algorithm and theoretical framework will broadly inform the theory and practice of field-based hexahedral meshing. By working in the space of globally meshable singular graphs and correcting octahedral fields to conform to this restricted set, we can circumvent debilitating degeneracies later in the meshing pipeline.

Chapter 6

Hex Me If You Can

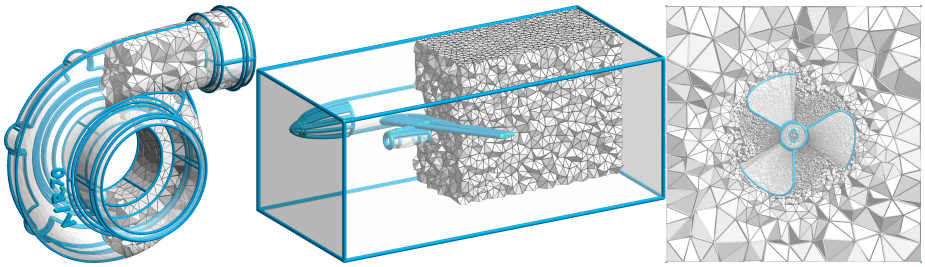


Figure 6.1: Three example models of HexMe (https://hexme.algohex.eu): The tetrahedral meshes faithfully represent feature points, curves (depicted in blue), and surfaces of the underlying CAD primitives.

Most of the hex meshing algorithms tackling the challenge use tetrahedral meshes as input or during some intermediate steps. However, there is no suitable tetrahedral dataset for objective analysis and meaningful comparison of hexahedral mesh generators. Our goal is to provide such a dataset to reveal common robustness issues and guide future research toward practical relevance.

The result of our endeavour is the HexMe dataset, a collection of tetrahedral meshes with tagged feature entities. The feature entities are special points, curves and/or surfaces, which need to be accurately captured by a hexahedral mesh, c.f. Fig.6.3. Please note that such feature points/curves/surfaces are common in all mesh generation scenarios, where they impose corresponding constraints on the local structure of the desired mesh. All meshes have been generated from computer-aided design (CAD) models, following a workflow defined with Snakemake[MJL+21], using the Gmsh[GR09] API with custom parameters defined in yaml metadata files. CAD models are classified into three categories (simple, nasty, industrial), in order to grade their difficulty and consistency. For each model, three meshes are provided: two resolutions (curvature-adapted, uniform) to analyze the mesh dependency of algorithms, and an embedding of the object into a box resulting in interior feature structures. The meshes including the feature tags are exported as `.vtk` datafiles (version 2, ASCII mode), a mesh format which is broadly used and easily accessible.

HexMe and HexaLab share the goal of guiding future research on hexahe-

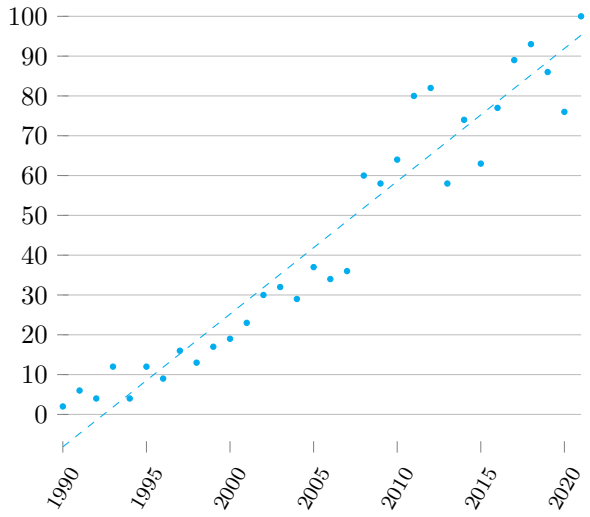


Figure 6.2: Yearly number of publications related to hexahedral meshing.
Source: app.dimension.ai (criteria: hexahedral mesh, in title and abstract)

dral meshing algorithms – HexMe through a suitable dataset of *input* models, HexaLab through the analysis and comparison of *output* hexahedral meshes.

The HexMe dataset has been designed to meet the following goals:

($\mathcal{G}0$) **Unambiguous & Ready-to-use** – The dataset offers volumetric tetrahedral meshes, ready to be used by any method relying on a tet mesh. Different methods can be compared without any bias introduced by ambiguous conversion procedures as for instance from surface or CAD representations.

($\mathcal{G}1$) **Challenging** – The dataset specifically includes *nasty* geometric configurations that are often avoided, as well as real-world *industrial* models containing an assembly of such difficulties.

($\mathcal{G}2$) **Discriminative** – The dataset allows to diagnose and grade the limitations of hexahedral algorithms. There are *simple* models enabling a sanity check, whereas each *nasty* model is designed to reveal robustness w.r.t. a specific type of difficulty. The tessellation dependence of a method is evaluated by including two different tetrahedral meshes for each model.

($\mathcal{G}3$) **Realistic** – The dataset mimics the workflow of a numerical practitioner, where all feature entities have to be preserved through all stages of the mesh generation pipeline. Explicitly defining such constraints is essential for comparison of methods since often it is possible to significantly improve the mesh quality or simplify the meshing task by violating some feature constraints. Consequently, output statistics are only comparable and meaningful if identical constraints have been enforced in the mesh generation.

($\mathcal{G}4$) **General** – So far, most hex meshing methods have been evaluated only on individual objects with a single boundary. This is only a special case of

the more challenging general volumetric meshing problem with arbitrary interior structures that arise for instance when simulating multi-material scenarios like fluid-structure interaction. Consequently, we include general test cases with interior feature constraints by embedding CAD models in a bounding *box*.

($\mathcal{G}5$) **Sustainable** – We intentionally limit the number of models to restrict the evaluation to challenging and meaningful geometries. This is important since hex meshing algorithm often include costly mixed-integer optimization and evaluating with thousands or millions of complex inputs is computationally infeasible. Moreover, the workflow has been designed in a sustainable manner, such that changes of the workflow require solely the regeneration of the affected models (Snakemake cache).

($\mathcal{G}6$) **Mutable** – Simultaneously to the algorithms, the set of challenging and meaningful geometries will change over time. HexMe is designed in a way that new models and even meshing definitions can be added conveniently (Snakemake workflow available on a public GitHub). We envision that the entire mesh generation community will actively contribute to future versions of HexMe.

In the following, we first describe in Section 6.1 datasets related to HexMe and discuss their differences. Afterwards, Section 6.2 introduces the pipeline to produce the tetrahedral meshes from CAD models, and Section 6.3 summarizes the content of HexMe. Finally, in Section 7.6, a state-of-the-art frame-field-based hexahedral meshing pipeline is applied to the dataset to verify that HexMe is suitable to benchmark robustness of hex meshing algorithms.

Dataset	Database	Mesher	Mesh Tri / Tet	Format	Features 0D—1D—2D—3D	Interface
Thingi10k	Thingiverse	-	Tri	`.stl`	×—×— ✓—×	query engine
TetWild	Thingi10k	TetWild	Tet	`.msh2`	×—×—×—×	Google drive
ABC	Onshape	Gmsh	Tri	`.obj`	×—✓—✓—×	chunks
SimJEB	GrabCAD	HyperMesh©	Tet	`.vtk`	×—×—×—×	webpage
[GSP19]	[FBL16] Drexel cad repository	-	Tri	`.obj`	✓— ✓— ✓— ×	`.zip`
HexMe	ABC GrabCAD MAMBO (crafted)	Gmsh	Tet	`.vtk`	✓—✓—✓—✓	web catalog

Table 6.1: Summary of datasets related to HexMe.

6.1 Related Datatsets

There is a shift among the scientific community. Open science that is readily reproducible and shared is becoming popular. Among other things, this is mainly

possible thanks to the free access to datasets supporting this open research. Consequently, an increased number of datasets has been published recently, where in the area of computer graphics ShapeNet [CFG+15], ModelNet[WSK+15], and Fusion360 [WPL+21] are popular resources. In the following, we present the five datasets in more detail that are most related to HexMe and discuss their similarities and differences.

Tetwild Even though Tetwild[HZG+18b] is a tetrahedral meshing technique, it is also a tetrahedral dataset, since the authors provide the output of their algorithm applied to Thingi10k[ZJ16], a triangular dataset. This tetrahedral dataset is the tetrahedrization of ten thousand models from Thingi10k. The tetrahedral meshes are `msh2` binary files, with a scalar per tetrahedron exposing the minimal dihedral angle. The 10k meshes are stored on Google Drive, within an archive `tar.gz` (∼9.5GB).

ABC The ABC[KMJ+19] dataset is a collection of one million computer-aided design models for geometric deep learning. All CAD files are from Onshape, and the original information related to those models is recorded within a metadata file *meta*`.yml`. Some processing tasks are done in order to filter the duplicate and broken models. The filtered models *para*`.zip` are afterwards converted into *step*`.step` and *stl2*`.stl` files using Parasolid. Gmsh[GR09] is then used to provide higher quality triangular meshes (either uniform, or adapted to the curvature), which are exported as *obj*`.obj` meshes from the `.step` files. Differential quantities are stored in those *obj* files, while the vertices and triangles of the mesh are respectively matched to the feature curves and patches, through another metadata file *feat*`.yml`. Further files may be provided, depending on the success of the processing. The dataset is downloadable by chunks containing `7z` archives of above files.

Thingi10k Historically, Thingi10k[ZJ16] is the first dataset providing ten thousand diverse, complex and quality `.stl` triangulations of 3D (printing) models. All models come from Thingiverse, and have been selected only if they were tagged *featured* by the Thingiverse staff. An online query interface is provided, which returns all the contextual and original information related to a `.stl` triangulation. It is also possible to download the whole dataset as an archive `tar.gz` from Google Drive (∼9GB).

SimJEB The recent SimJEB[WBM21] dataset provides 381 tetrahedral meshes from CAD models, by following a *semi-automated* pipeline. The CAD models come from a challenge organized by GrabCAD. Those former 700 models have been filtered (mostly based on the filename), manually repaired, and then meshed using the commercial software HyperMesh. A structural simulation was performed using the commercial software OptiStruct. The 381 `.vtk` tetrahedral meshes surviving this pipeline are hosted through the Harvard Dataverse (∼1.6GB), along with the corresponding clean CAD `.stp` file, triangular surface `.obj` meshes, finite element `.fem` models, and simulation `.csv` results. The final models are identified by an integer, specified by *readme* files. A web page allows to browse the designs, and to explore the data.

[GSP19] Recent grid-based hexahedral meshing papers [GLYL20, PLC+21, LPC21] have benchmarked their method on a common dataset gathered by [GSP19]. This dataset consists of 202 `.obj` triangular meshes from [FBL16] and Drexel CAD repository (2018), 109 of them are equipped with feature curves. The feature curves have been extracted using a dihedral angle of 140 degrees in combination with manual adjustment. A `.fgraph` file specifies the set of feature edges but without grouping them semantically into feature curves. The 202 triangular meshes with the 109 feature annotations are available as a supplementary `.zip` file (∼1.4GB).

The available datasets differ by (i) their selection of models, (ii) whether triangular or tetrahedral meshes are supplied, (iii) potential specification of feature entities (0D/1D/2D/3D), and (iv) their infrastucture to access models and to potentially contribute new models. Table 6.1 summarizes the comparison between HexMe and the related datasets. Please note that for datasets solely providing triangular surface meshes, we interpret these as a 2D feature specification, even if no explicit tags are available.

SimJEB is *by-design* the closest dataset to HexMe: small number of tetrahedral meshes from CAD models, semi-automated meshing pipeline, web catalog and `.vtk` mesh format. However, all SimJEB geometries describe a jet engine bracket, while HexMe supplies more diverse and challenging types of geometries. ABC is the only related dataset providing meshes with feature tags corresponding to the CAD geometry. However, the ABC dataset consists only of triangular surface meshes and does not define any volumetric discretization. The dataset of [GSP19] supplies 109 triangular surface meshes with annotated feature points and curves. But neither tetrahedral meshes, nor tags relating feature entities to the corresponding CAD primitives, or a sustainable and extensible workflow are available. The Thingi10k and Tetwild meshes have not been generated from CAD files and thus do not specify feature entities.

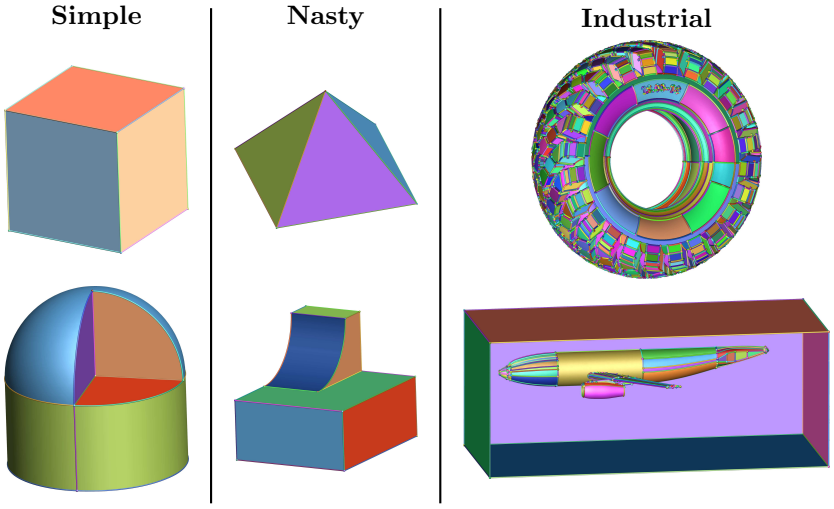


Figure 6.3: HexMe uses three categories of CAD models: simple, nasty and industrial.

6.2 From CAD to Tets

All the tetrahedral meshes provided by HexMe have been produced from three categories of CAD models, Figure 6.3:

- **s**imple models: basic shapes that are assumed to be easily meshable, i.e. the target hexahedral topology is fair, e.g., a cube (`s01o_cube.geo`), or a cut hemisphere on a cylinder (`s10o_cyl_cutsphere.stp`).
- **n**asty models: academic shapes that are challenging to hex-mesh, e.g., a pyramid[VPR19] (`n09o_pyramid.geo`), or a ski jump (`n02o_skijump_anti_box_cyl.geo`).
- **i**ndustrial models: lifelike shapes, which hexahedrization is highly valuable for numerical simulation, e.g., a truck tire (`i28o_gc_tire_1218.step` from GrabCAD), or an aircraft for CFD (`i31o_dlr_f6.brep` [VTM+08, BERF08]).

The 63 CAD models have been selected to cover known configurations that are challenging for current hexahedral meshing algorithms. In the future, the dataset will evolve together with the algorithms with the goal of providing a minimal set of test cases, which is maximally meaningful.

In contrast to the comparable datasets (Section 6.1), the CAD models come from several databases: ABC dataset (originally from Onshape), GrabCAD and MAMBO. Some of the CAD models were created specifically for HexMe, using Gmsh and Siemens NX software. Those latter models are released to the *public domain*, while the other ones are regulated by licenses, which are respectively: Onshape Terms 1(g)(i), GrabCAD Terms (c.f. related FAQ) and Apache 2.0. The above information is reported within a metadata file (with the following nomenclature `(s|n|i)(\d{2})o_{extra}.yaml`) per CAD model with a short description of the shape, e.g. `i28o_gc_tire_1218.yaml`:

```yaml
author: Milos Suvakov, grabcad.com/milos.suvakov
description: tire of a truck
license: GrabCAD Terms
name:  i28o_gc_tire_1218
original: original/i28o_gc_tire_1218.step
references: https://grabcad.com/library/tire-12-00-18-1
```

For each CAD model, three tetrahedral meshes are provided, c.f. Figure 6.4:

- **c**urvature-adapted: the mesh element size is adapted to the curvature, and upper bounded such that the CAD geometry is sufficiently preserved (e.g. `i05c_m5.vtk`).
- **u**niform: the mesh element size is constant, even in the neighborhood of the tiniest geometrical features (e.g. `i05u_m5.vtk`).
- **b**ox-embedded: the initial model is embedded in a box that is twice as large as the original bounding box, and the corresponding mesh is generated such that the smallest gap between the box limits and the initial model is meshed by one layer of tetrahedra (e.g. `i05b_m5.vtk`).

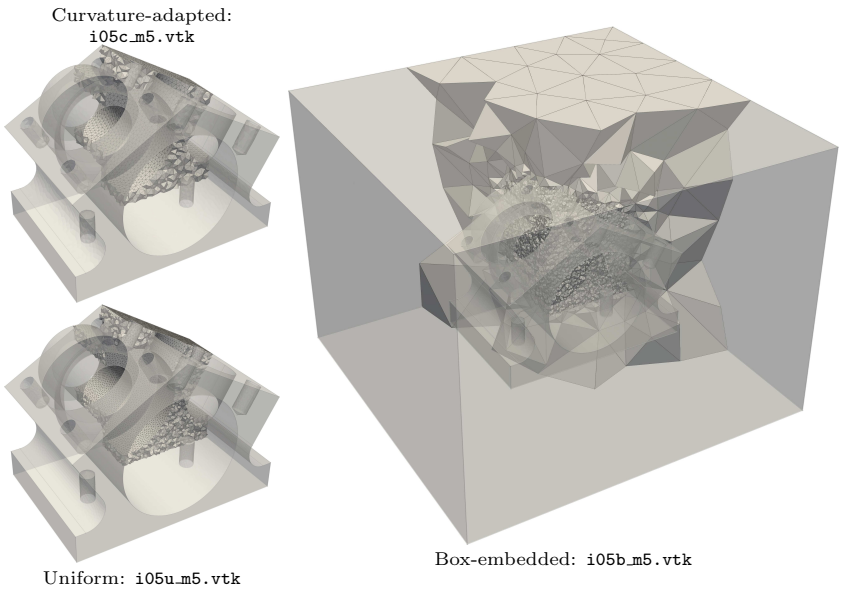


Figure 6.4: There are three tetrahedral meshes per CAD model, e.g. `i05o_m5.step` from MAMBO.

The pipeline for the mesh generation is orchestrated by Snakemake[MJL+21], a popular (currently ~ 7 citations per week) scalable workflow management system. In a few words, Snakemake is a modern version of GNU Make, whose syntax is close to Python. The workflow `Snakemesh` consists of two rules. The first rule `meshes` defines which meshes should be produced. The second rule `cad2vtk` generates a mesh from a CAD model and a metadata file `(s|n|i)(\d{2})(c|u|b)_{extra}.yaml` containing the custom mesh options (for **c**urvature-adapted, **u**niform, or **b**ox-embedded). To do so, this second rule runs a python script using Gmsh API, with a maximum of 8 threads. For each mesh, a log file `(s|n|i)(\d{2})(c|b|u)_{extra}.txt` is written with the corresponding console output, in order to record the history of the meshing task.

Snakemake scans the workflow in a backward fashion, meaning that the input files are determined from the output ones. In other words, the purpose of the first rule is to state all meshes that should be produced. Afterwards, the second rule provides those meshes by identifying the corresponding input files accordingly, which are the CAD model and the metadata file. This backward identification is the key of the workflow definition, since the rules are mostly written with wildcards. The use of Snakemake easily yields a sustainable dataset, since a rule is applied only if an output is either missing or older than the corresponding input.

Gmsh[GR09] does not only mesh the volume, but also the feature entities as defined by the CAD model. In addition to the tetrahedral elements, there are triangle, edge and vertex elements (lower dimensional elements are conforming to the higher ones) to respectively discretize feature surfaces, feature curves and feature points. Those features are identified by the CAD with a *tag* (i.e. a positive integer), which corresponds to a physical group within Gmsh. Doing so, the corresponding mesh elements are created accordingly with the inherited

CAD tag. Meshes are exported as vtk Datafile Version 2.0, in ASCII mode. The used Gmsh git-version is written within the file header. A mesh is defined as an `UNSTRUCTURED_GRID`, with the four following sections:

1. `POINTS`: coordinates of every node
2. `CELLS`: number of nodes and nodal definition of every element (vertices, edges, triangles and tetrahedra); the second number on the cell section is the total number of integer values
3. `CELL_TYPES`: integer corresponding to the element type {1:vertex, 3:edge, 5:triangle, 10:tetrahedron}
4. `CELL_DATA`: integer corresponding to the element tag that belongs to the CAD feature.

```
# vtk DataFile Version 2.0
s01o_cube, Created by Gmsh 4.9.3-git-ac3fcda9f
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 273 double
-0.9559524353885869 0.9350753925146784 1.325609826087499
[...]
CELLS 1425 6569
1 0
[...]
CELL_TYPES 1425
1
[...]
CELL_DATA 1425
SCALARS CellEntityIds int 1
LOOKUP_TABLE default
1
[...]
```

Overall, there are 189 meshes, whose filenames follow the nomenclature `(s|n|i)(\d{2})(c|u|b)_{extra}.vtk`, which summarizes the corresponding model `(s|n|i)(\d{2})` and mesh `(c|u|b)` types.

6.3 HexMe Anatomy

The HexMe tetrahedral dataset is downloadable in a single file: hexme.zip (∼1.5GB). Alternatively, it is possible to download meshes *one-by-one* from the catalog (the catalog is mostly generated by Snakemake, using the report feature) The catalog is split into three categories (i, n, s), that correspond to the model categories (respectively: **i**ndustrial, **n**asty, **s**imple). Within each category, there are three subcategories (b, c, u), that correspond to the mesh types (respectively: **b**ox-embedded, **c**urvature-adapted, **u**niform).

An entry of the catalog is described by two pictures (a cut view and a quality histogram), a `.pdf` file, a `.vtk` mesh, the corresponding log file `(s|n|i)(\d{2})(c|u|b)_{extra}.txt` and the metadata file `(s|n|i)(\d{2})o_{extra}.yaml` related to the CAD model. A summary of the mesh is also available in a `.pdf`

sheet, Figure 6.5. The summary provides topological information about the CAD model (number of points, curves and surfaces) and the mesh (number of vertices, edges, triangles, tetrahedra and nodes). Moreover, two histograms related to the inverse condition number (ICN)[JGTR16, §2.1] of triangles and tetrahedra are plotted. Finally, four screenshots (xy-, yz-, zx- and 3D-views) of the cut mesh are displayed.

On top of HexMe catalog, there is a GitHub page hosting all the necessary input files to run the workflow. The tetrahedral meshes are not hosted on this git repository (the git history would be too heavy otherwise). The main purpose of this git repository is to expose the workflow that has been used for the mesh generation. Through this repository, it is possible to report an `issue` if a mesh does not meet user expectations. The meshing community is invited to actively contribute to the HexMe dataset, by creating `pull-requests` for proposing new models and/or filtering of existing ones. There will be releases with appropriate `git-tag`, whenever the dataset has been significantly updated.

How to contribute. The workflow supports CAD models with extensions `.geo`, `.step`, `.stp`, or `.brep`. For each CAD model, four metadata files need to be defined. The first one specifies general information about the model

```yaml
# meta/(i|n|s){\d{2}}o_{extra}.yaml
description: ...
license: ...
name: (i|n|s){\d{2}}o_{extra}
original: original/(i|n|s){\d{2}}o_{extra}.(geo|stp|brep|step)
references: ...
```

The other three files define the desired meshing parameters for each mesh type `(c|u|b)`, respectively:

```yaml
# meta/(i|n|s){\d{2}}c_{extra}.yaml
gmsh.option.setNumber:
- Mesh.Algorithm: ...
- Mesh.Algorithm3D: ...
- General.NumThreads: ...
- Mesh.CharacteristicLengthMax: ...
- Mesh.MeshSizeFromCurvature: ...
info: meta/(i|n|s){\d{2}}o_{extra}.yaml
```

```yaml
# meta/(i|n|s){\d{2}}u_{extra}.yaml
gmsh.option.setNumber:
- Mesh.Algorithm: ...
- Mesh.Algorithm3D: ...
- General.NumThreads: ...
- Mesh.CharacteristicLengthMin: ...
- Mesh.CharacteristicLengthMax: ...
info: meta/(i|n|s){\d{2}}o_{extra}.yaml
```

```yaml
# meta/(i|n|s){\d{2}}b_{extra}.yaml
gmsh.model.mesh.setSize:
- ipts: ...
gmsh.option.setNumber:
- Mesh.Algorithm: ...
- Mesh.Algorithm3D: ...
```

```
- General.NumThreads: ...
- Mesh.MeshSizeExtendFromBoundary: ...
- Mesh.MeshSizeFromPoints: ...
- Mesh.MeshSizeFromCurvature: ...
info: meta/(i|n|s){\d{2}}o_{extra}.yaml
```

The items of the section `gmsh.option.setNumber` correspond to the options that Gmsh provides. The items `Mesh.CharacteristicLength*` constrain the mesh element size range. In the case of a uniform mesh, those values are chosen identically. For further details about Gmsh options, we refer the reader to the corresponding documentation. The `info` entry links the mesh metadata to the metadata of the CAD model. Finally, observe that the box-embedded metadata has a section related to `gmsh.model.mesh.setSize`. The value of `ipts` is the mesh element size used for the interior of the CAD model (we recommend to use the element size of the uniform meshing case by default). For such box-embedded meshes, the outer part of the model is meshed with one layer of tetrahedra in the smallest gap between the model boundary and the limits of the box.

6.4 Example Evaluation

To demonstrate the value of HexMe, we challenge the *state-of-the-art* pipeline of frame-field based hexahedral meshing described in Chapter 1 with our dataset and evaluate its robustness. Robustness issues regularly stem from hard constraints induced by feature points, curves and surfaces. Consequently, our primary concern here is to quantify how faithful the tagged input features are reproduced in the generated hexahedral meshes. Note that this is a novel way to assess robustness, which is more fine-grained than simply counting *passed or failed* per model. It is also more meaningful in our setting than other straightforward choices (percentage of hexahedral elements, or distortion of the integer-grid map for instance) where excellent numbers could be reported despite some feature constraints are violated.

For the same reason, typical quality metrics (such as the scaled Jacobian) have been omitted since they are meaningless for state-of-the-art algorithms, where most of the generated hexahedral meshes are incomplete. Obtaining a high quality hexahedral mesh of a subregion is often significantly easier if features are not preserved. However, there is no doubt that quality metrics will be important in the future of HexMe as soon as the robustness of available algorithms reaches a sufficient level.

The evaluation of the standard frame-field based hex-meshing pipeline consists of the following steps:

1. Determination of the target edge length h
2. Specification of frame-field alignment constraints.
3. Feature-aligned smooth frame-field generation. [RSL16]
4. Integer-Grid map generation guided by the frame-field. [NRP11]
5. Hexahedral mesh extraction from the IG map. [LBK16]

HEXME – i08c_m8

	# Points	# Lines	# Surfaces
CAD Features	100	148	52

	# Vertices	# Edges	# Triangles	# Tetrahedra	# Nodes
Mesh Elements	100	3655	112942	358605	87143

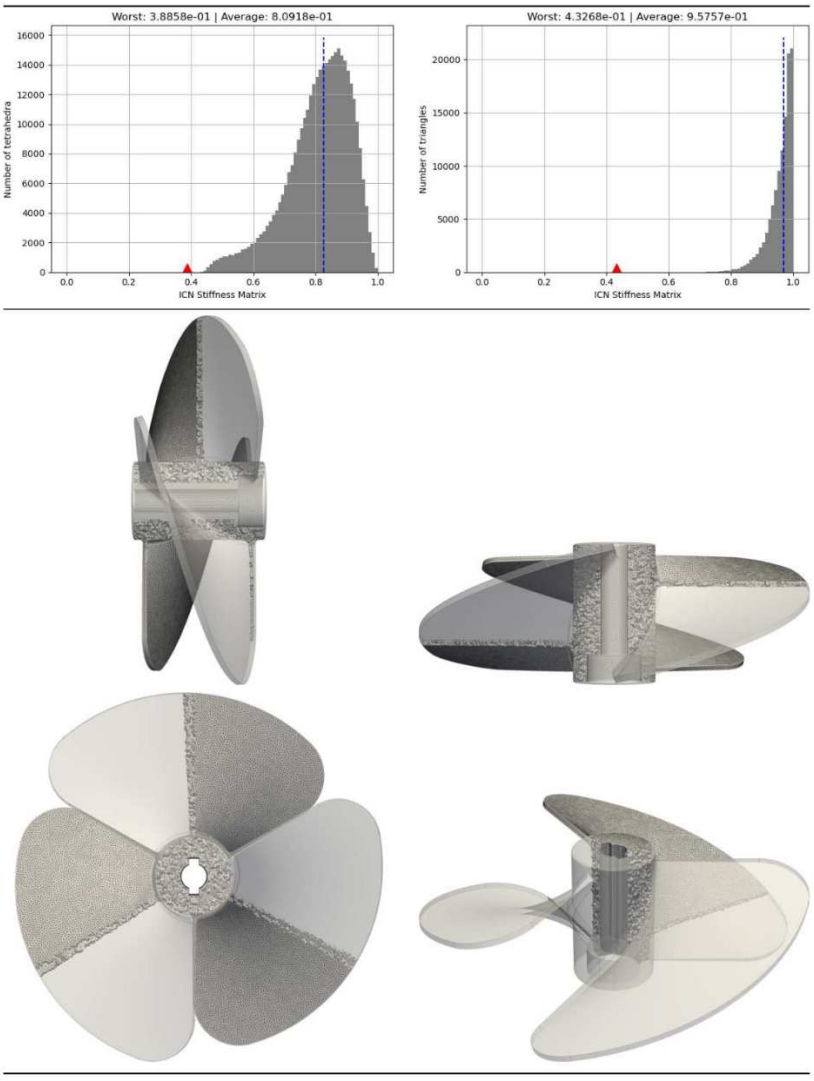


Figure 6.5: Sheet summarizing `i08c_m8`.

6. Verification of feature points, curves, and surfaces.

While upon success the above algorithm delivers promising hexmesh qual-

ity, it can neither guarantee a valid integer-grid map, nor a valid hexahedral mesh. Failures are caused by (i) non-meshable frame-field topologies or (ii) the inability to guarantee local injectivity for volumetric maps, see [PCS+22] for more details. Since all of the above defects are frequently triggered by feature constraints, HexMe is well-suited to quantify the lack of robustness of the candidate pipeline. Please note that we did not include singularity repair strategies based on collapses [LLX+12] or splits [JHW+14], since their repair capabilites are comparable to what HexEx [LBK16] is able to handle throughout the extraction step. The above hex-meshing pipeline is released as an open-source C++ library to enable reproduction and ease comparison for future research.

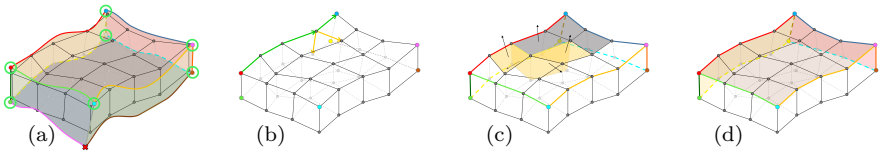


Figure 6.6: Feature matching process: feature points (a), feature curves (b), feature patches (c), and the final result (d).

Before presenting and discussing the results of our evaluation, we describe in more detail the choice of the target edge length (Step 1), frame-field alignment constraints (Step 2), and the verification of feature entities (Step 6). The verification of represented features requires special attention since the integer-grid map might contain degeneracies that are repaired by HexEx but prevent a trivial transfer of feature tags from the tetrahedral to the hexahedral mesh.

Target Edge Length. We determine the target edge length h of the hexahedral mesh such that n hexahedra are generated. Considering that the volume of the unit cube is one, we obtain $h = \left(\frac{V}{n}\right)^{1/3}$, with V being the volume of the input tetrahedral mesh. For all our experiments we choose $n = 50k$.

Frame-Field Alignment Constraints. Frame-fields can be seen as a continuous relaxation of an integer-grid map, cf.[PCS+22]. A frame exhibits the same symmetries as a cube and therefore represents the local rotation of a cube. Consequently, we require that the frames align tangentially to all feature curves and feature surfaces. In a HexMe tet mesh, a feature curve is represented by a 1-manifold chain of edges. For each of the interior vertices along such a chain, we estimate a tangent vector by averaging the two incident edges and then use it as an alignment constraint for the frame-field. Similarly, a feature surface is represented by a 2-manifold subset of triangles and we compute (area-weighted) vertex normals that are also used as alignment constraints.

Verification of Feature Points. For each feature point of the input tet mesh, we search the closest vertex in the hex mesh. If the distance is below $\tau = h$, the feature point is counted as correctly reproduced, otherwise as invalid, as illustrated in Fig. 6.6 (a). Please note that we do not require higher-accuracy geometric re-produciton of features but merely want to assess whether the connectivity of the hex mesh is able to represent the feature.

Verification of Feature Curves. In HexMe, a feature curve always contains a feature point. Consequently, we can verify feature curves by a greedy path search from a source feature vertex to a target feature vertex in the hexahedral mesh. At the source vertex we begin with the edge most parallel to the feature curve tangent at the corresponding point in the tetrahedral mesh. Then we extend the path through subsequent vertices by always following the next edge, which is most parallel to the previous one, as illustrated in Fig. 6.6 (b). This simple geometric heuristic is motivated by the fact that feature curves represent smooth 1-manifolds, and turned out to be reliable in our experiments. A feature curve is successfully verified if the path reaches the target vertex, while all intermediate points have a distance below τ. In all other cases, we classify the feature curve as not being reproduced correctly. Please note that whenever the source or target feature point are not verified in the previous stage, the feature curve verification will automatically fail.

Verification of Feature Surfaces. For verification of feature surfaces we use a geometrically guided strategy similar to the one for feature curves. Starting from a seed quadrilateral, we grow the surface in a breadth-first manner to neighboring elements based on normal similarity, as illustrated in Fig. 6.6 (c). We ensure 2-manifoldness by disallowing growth that would result in more than 2 incident quadrilaterals at one edges. The growth procedure terminates when newly added elements reach a distance above τ to the input feature surface, or when it encounters edges belonging to feature curves. The resulting surface is correctly reproduced if its boundary feature curves coincide with those specified in the input tetrahedral mesh. Again, a feature surface can only be correctly reproduced, if all its incident feature curves are already correctly reproduced.

Results and Discussion. We run the hexahedral mesh generation pipeline on all HexMe models and count the percentages of correctly reproduced feature points, curves, and surfaces. A table of full statistics can be found in the supplement. Only for 19 out of 189 models all feature entities are reproduced correctly, while the average success rates are 70.9% / 48.5% / 34.6% for feature points/curves/surfaces.

Figure 6.7 (a) shows the cumulative percentage of simple/nasty/industrial models where *at least* a certain percentage of feature points/curves/surfaces in the tetrahedral mesh (of any type, box-embedded/curvature-adapted/uniform) are present in the corresponding hexahedral mesh. As expected, the hexahedral pipeline is performing the best on the category of simple models. Around 50% of the hexahedral meshes preserve at least 100% / 86% / 63% of the feature points/curves/surfaces of the input tetrahedral meshes. For the meshes of the nasty models, those figures drop to 100% / 34% / 1%, while for the meshes of the industrial models they are only 78% / 32% / 8%.

Figure 6.7 (b) provides similar information, but for the curvature-adapted, uniform and box-embedded meshes of all models. In the case of the box-embedded meshes, the feature verification does not count the features corresponding to the box (8 points, 12 curves, 6 surfaces), since those features are trivially recovered. The hexahedral pipeline performs better for uniform meshes than for curvature-adapted. While integer-grid map based approaches do not necessarily require a dense tetrahedral mesh in their domain, they explicitly

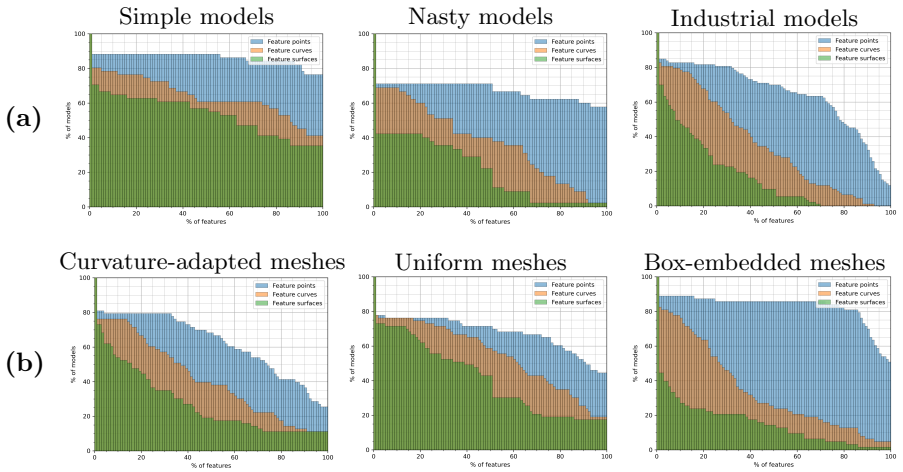


Figure 6.7: Cumulative histograms of the percentage of (a) models and (b) meshes, in respect to the percentage of tetrahedral features that are recovered in the hexahedral meshes. Observe that in the case of the box-embedded mesh type, the features of the embedding box (8 points, 12 curves and 6 surfaces) are discarded.

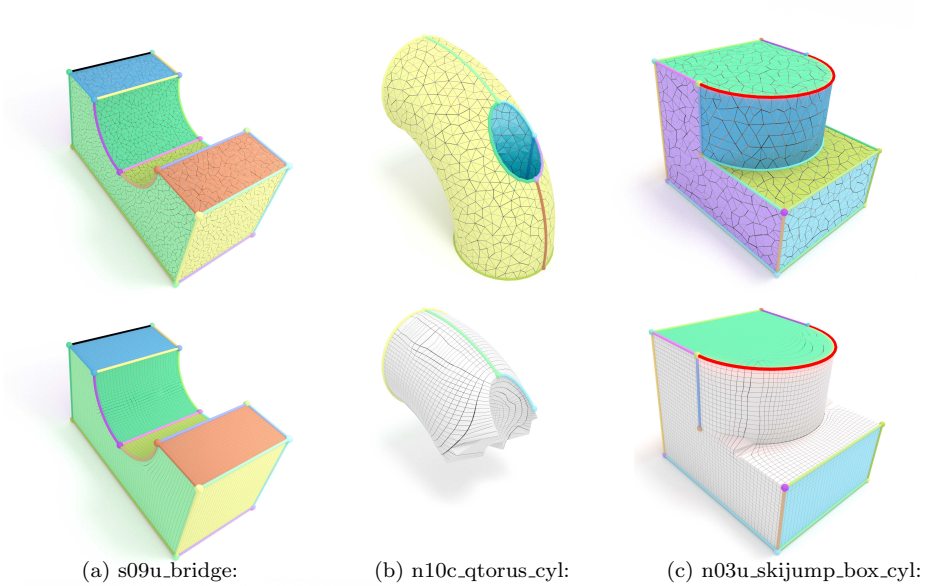
require mesh vertices and edges to represent singularities. Locally inadequately coarse meshes can therefore lead to collapses in the singularity structure that induce global non-meshability and consequently worsen feature reproduction. The inner features of the box-embedded meshes are clearly more challenging to preserve compared to the uniform and curvature-adapted meshes.

Figure 6.8 visualizes five representative examples of HexMe tet meshes and their corresponding generated hex meshes. Feature points, curves and surfaces are color-coded in the tet meshes. The corresponding feature entities in the hexahedral mesh are only color-coded if reproduced correctly. Here we solely provide general statistics and some qualitative examples. A complete enumeration and categorization of defects are out of the scope of this evaluation but will be an important task for future work.

6.5 Conclusion

The contributions of HexMe are twofold. On the one hand, it is a tetrahedral dataset with tagged feature entities and on the other hand, it is a transparent workflow. The main objective of HexMe is to provide tetrahedral meshes for the meaningful assessment of hexahedral meshers and associated auxiliary tools such as 3D frame-fields. In the future, the choice of meshes will evolve together with the progress of hexahedral meshing techniques. Therefore, we publish the full workflow to ensure that the HexMe dataset can be updated easily and does not become outdated in the future.

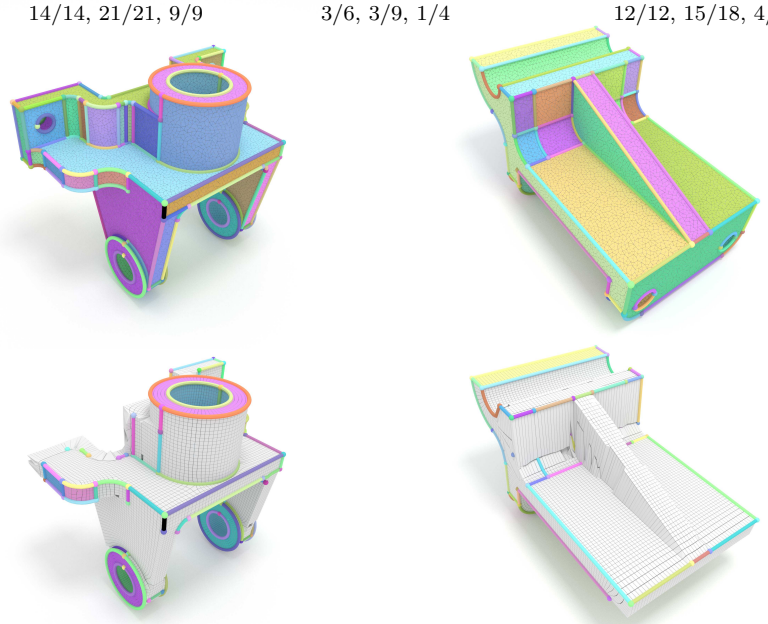
The selected 63 CAD models come from several databases. Their origin and license are recorded within a metadata file. There are three categories of CAD models, and three types of meshes per CAD model. The 189 meshes



(a) s09u_bridge: 14/14, 21/21, 9/9

(b) n10c_qtorus_cyl: 3/6, 3/9, 1/4

(c) n03u_skijump_box_cyl: 12/12, 15/18, 4/8



(d) i03u_m3: 89/111, 89/169, 13/60

(e) i06u_m6: 52/68, 59/105, 7/39

Figure 6.8: Example input tet meshes (left) and output hex meshes (right). The numbers indicate ratios of correctly reproduced feature points, curves and surfaces, as additionally depicted visually.

are produced thanks to a workflow that is defined with Snakemake. The CAD features are reproduced by Gmsh as lower dimensional elements (vertices, edges, triangles), with corresponding tags. The meshes are expressed as `.vtk` Datafile Version 2.0 in ASCII mode.

There are two ways to access the HexMe tetrahedral meshes – either by downloading all of them in a 1.5GB `.zip` file, or by picking individual ones from the catalog. In addition to the meshes and log files, the catalog yields the metadata related to the CAD model, a summary about the mesh, and information related to the workflow. The files that are involved in the workflow, are available on a GitHub repository. From this git repository, it is possible to raise issues and/or pull requests to improve the dataset or the workflow.

The commit corresponding to the version used for this paper has been tagged `HexMe-1.0`. The dataset has been uploaded to Zenodo, and can be referenced with the following `doi` 10.5281/zenodo.6642020. Whenever a new release occurs, those version tags will be updated accordingly. Those taggings are crucial in order to keep track of the assessment of hexahedral methods.

Chapter 7

Locally Meshable Frame Fields

This chapter mainly targets the robustness issue of the automatic field based hex meshing pipeline, particularly non-meshable topological configurations that exist in volumetric frame fields and are frequently generated by state-of-the-art algorithms [RCR19]. In Sec. 7.1.2, we carefully analyze the difference between topological structures that exist in general 2D frame fields but are not possible in quad mesh-induced frame fields. Necessary and sufficient conditions for the meshability of a frame field are identified, and we discuss an algorithm to turn a given frame field into a locally meshable, as well as further processing into a (globally) meshable one. Sec. 7.2 is devoted to the meshability of 3D frame fields. We carefully analyze conditions for the local meshability of singular arcs and singular nodes and study their decomposability into fundamental pieces that can be handled algorithmically. With the help of *arc zipping*, we design an algorithm that converts an input frame field into a locally meshable one. In Sec. 7.3 and 7.4, we extend the conceptual ideas to a concrete implementation based on piecewise constant frame fields on tetrahedral domains. In Sec. 7.6, we evaluate our novel algorithm on the challenging HexMe dataset [BRK+22]. The results show that locally meshable frame fields achieve significantly higher success rates in terms of valid integer-grid maps than state-of-the-art frame field based methods. We also present a novel algorithm in Sec. 7.4 to optimize seamless maps and integer-grid maps for a given frame field, further improving the robustness. An overview of the novel hex meshing pipeline is demonstrated in Fig.7.1.

7.1 Meshability in 2D

7.1.1 2D Frame Field Topology

The topology of a 2D frame field is fully characterized by the behavior of streamlines of its corresponding vector field on a 4-sheeted branched cover of the 2D domain, as introduced in [KNP07b]. Consequently, we will briefly revisit vector field topology before proceeding with frame field topology. For a detailed and complete discussion of vector field topology, we refer the reader to

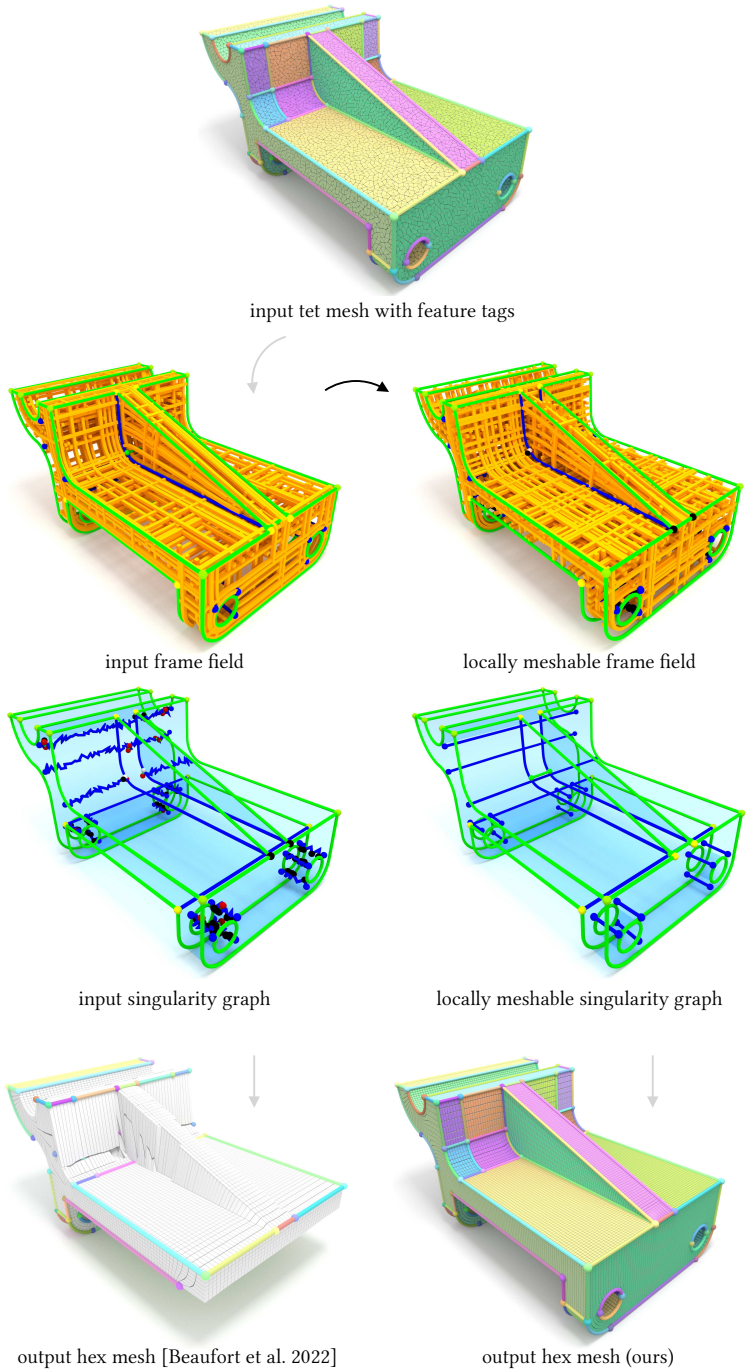
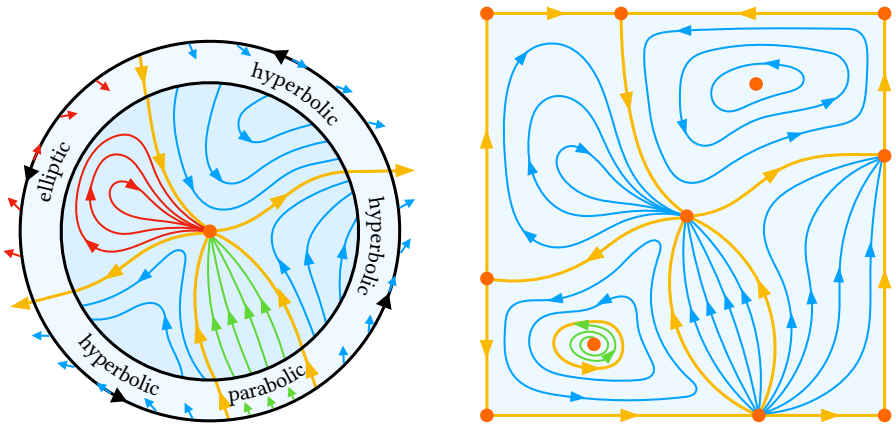


Figure 7.1: Overview. Non-meshable topological configurations in frame fields, e.g., invalid singularities or feature structures, induce degenerate integer-grid maps and broken and incomplete hex meshes (top row). Our algorithm (bottom row) automatically turns a given frame field into a locally meshable one, where a valid integer-grid map enables a hex mesh that preserves all input features.

[Asi93, GBR21].



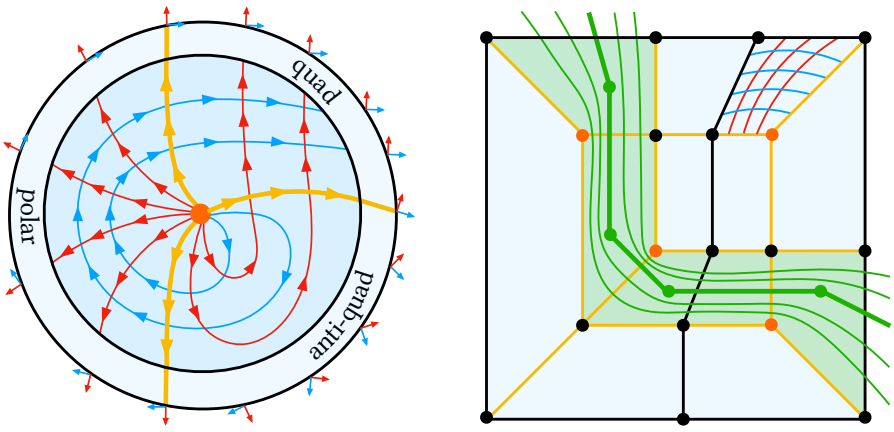
(a) (b)

Figure 7.2: (a) Local vector field topology of an isolated singularity (orange) with five sectors, formed by streamlines of parabolic(green), hyperbolic (blue), and elliptic(red) flow behavior. Separatrices (yellow) are either parabolic or elliptic streamlines that divide space into sectors of different flow behavior. (b) Global vector field topology: The topological skeleton consists of all separatrices (yellow), dividing space into regions of identical flow behavior, potentially including closed orbits. A limit cycle (green) is a streamline that converges to a closed orbit.

Local Vector Field Topology An important topological feature of a vector field $v(x) : \Omega \to \mathbb{R}^2$ are its singularities, i.e. points, curves or regions where $||v(x)||_2 = 0$. In our setting, only isolated point singularities are of importance, where the vector field does not vanish in an ϵ-disk neighborhood D_ϵ of the singularity with circular boundary $\partial D_\epsilon = C_\epsilon$. In such a local neighborhood of a singularity, there are only four different types of streamlines that can be distinguished based on their limit behavior in forward and reverse flow directions, as illustrated in Fig. 7.2. A streamline is called a *closed orbit* if it cycles on a closed path around the singularity, which in this case is called a *center*. Otherwise, the flow in a forward/reverse direction can either converge to the singularity or diverge away from it. A *parabolic* streamline converges in one and diverges in the other direction. It is called *inflow* if the forward direction is converging; otherwise, it is called *outflow*. A *hyperbolic* streamline diverges in both directions, and an *elliptic* streamline converges in both directions. Partitioning all streamlines passing through the local neighborhood circle C_ϵ according to their type results in a cyclic sequence of parabolic, hyperbolic, and elliptic sectors. This cyclic sequence fully specifies the topological type of a point singularity. A *separatrix* is a parabolic or elliptic streamline separating the space between two sectors of different flow behavior. An example of the local vector field topology at an isolated singularity is shown in Fig. 7.2a. The *index* of a singularity quantifies how many full rotations the vector field undergoes when traversing C_ϵ in the counter-clockwise sense. Considering that parabolic sectors are bounded by separatrices of identical type, i.e., either both inflow or both outflow, while

hyperbolic and elliptic sectors are bounded by separatrices of different flow direction, the integer-valued index of a singularity with n_p parabolic, n_h hyperbolic, and n_e elliptic sectors can be computed by $index = 1 - \frac{1}{2}n_h + \frac{1}{2}n_e$. The signs reflect the fact that in hyperbolic sectors, the transition from inflow to outflow is a clockwise rotation, while in elliptic sectors, it is counter-clockwise. Due to the transitional in $\leftrightarrow$ out flow behavior of hyperbolic and elliptic sectors in combination with the continuity of the vector field, $n_h + n_e$ is even at non-boundary points, ensuring that the index is an integer as expected.

Global Vector Field Topology The *topological skeleton* encodes the global topology of a vector field by partitioning the domain into regions of identical asymptotic flow behavior. It consists of the union of all singularities and separatrices in the entire domain, including those induced by singularities and, moreover, closed orbits at the interface between different flow behavior. On the global scale, there is one additional type of asymptotic streamline behavior that cannot be observed near isolated singularities. A streamline is called a *limit cycle* if it asymptotically converges to a closed orbit. Fig. 7.2b depicts an example of a topological skeleton including a limit cycle (green).



(a) (b)

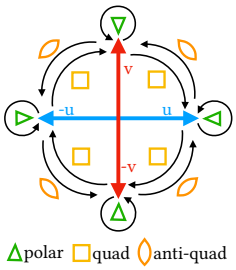
Figure 7.3: (a) Local frame field topology at an isolated singularity (orange) with three different sectors. Frame field sectors can be of quad, polar, or anti-quad type. Only quad sectors are meshable since the other two generate triangular patterns and even a digon in the anti-quad case. (b) Mesh-induced frame field.

Local Frame Field Topology A *frame* consists of four $\mathbb{R}^2$ vectors $\{u, v, -u, -v\}$, which are pairwise anti-parallel and often specified by a matrix $F = [u, v] \in \mathbb{R}^{2\times 2}$. A *frame field* corresponds to a vector field on a 4-sheeted branched cover, where pointwise each branch corresponds to one of $\{u, v, -u, -v\}$, and the connection is induced by the covering space. For non-singular points with $||F||_2 \neq 0$, we require $\det F > 0$ such that u-streamlines can never be parallel to v-streamlines when projected onto the domain, and a u-vector on one branch cannot vanish independently of the v-vector on the other branch.

It would be possible to solely discuss the topology of a frame field in terms of the topology of the vector field on the 4-sheeted branched cover. However,

topological sectors will overlap when projected to the original domain, and several topological configurations that would exist for entirely independent vector fields are excluded by the $\det F > 0$ condition. This motivates the definition of new types of frame field sectors, which are meaningful to the domain itself and fully characterize the frame field behavior. Similarly to the vector field case, topological sectors at a singular point are defined by subsequent separatrices and the rotational behavior of the field in between them. However, in the case of a frame field, subsequent separatrices might belong to different branches, e.g., an outflow-u-separatrix might be followed by an outflow-v-separatrix.

Due to the continuity of the frame field, when expressed in a common coordinate chart, two subsequent separatrices can only be on (i) the same branch, (ii) the next branch, or (iii) the previous branch, e.g. an outflow-u separatrix can only be followed by an outflow-u, an outflow-v, or an inflow-v separatrix. We call the corresponding three types of sectors *polar*, *quad*, and *anti-quad*, and show examples in Fig. 7.3a. The index of a frame field singularity with n_q quad, n_p polar, and n_a anti-quad sectors can be computed by $index = 1 - \frac{1}{4}n_q + \frac{1}{4}n_a$, which is a quarter-integer from $\frac{1}{4}\mathbb{Z}$.



Global Frame Field Topology In analogy to the vector field case, the topological skeleton consists of the union of all singularities and separatrices, which partition the domain into flow-aligned regions. In contrast to the vector field case, additional nodes of the topological skeleton are induced by the intersection of separatrices on different branches.

7.1.2 2D Frame Field Meshability

We call a 2D frame field *meshable* if a quadrilateral mesh of identical topology exists. Comparing the topology of a frame field with the topology of a mesh is possible through the notion of the *mesh-induced frame field topology* introduced next.

Mesh-induced Frame Field Topology The edges of a quadrilateral mesh in 2D can be interpreted as a discrete representation of the streamlines of an underlying frame field, as illustrated in Fig. 7.3b. A discrete streamline following a mesh edge continues through a regular vertex, and either forms a closed cycle or terminates at a boundary point or a singular vertex. The topology of all remaining streamlines through the domain is uniquely determined by the flow along edges of the dual mesh in the following way: All streamlines in-between two opposite edges of a quadrilateral are topologically identical to the corresponding dual streamline, i.e. the channel formed by a strip of subsequent quads, as illustrated in Fig. 7.3b. Please note that we have fully defined the topology of the mesh-induced frame field without explicitly constructing or defining the geometry of the field.

Meshability Conditions Comparing the topological structures that exist in general frame fields with those induced by a mesh, we deduce the following necessary and sufficient conditions for meshability of a frame field in 2D:

(C1) *Feature Alignment* – The frame field aligns to all feature curves on the domain to ensure their correct re-production in the quad mesh.

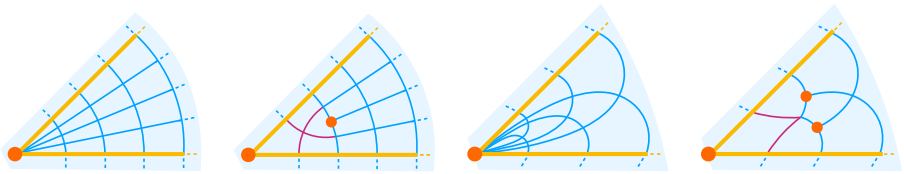
(C2) *Isolated Singularities* – All singularities with $||F||_2 = 0$ are isolated points, and there are no curve or patch singularities, which might cause mesh degeneracies.

(C3) *Quad sectors only* – The frame field does not contain any polar or anti-quad sectors, which would induce non-quad cells in the mesh.

(C4) *No limit cycles* – A meshable frame field cannot contain limit cycles since all streamlines of the mesh-induced frame field topology are either closed orbits or end at a singularity or the boundary.

The sufficiency of the conditions can be verified in a constructive way. If (C1)-(C4) are satisfied, all cells of the topological skeleton are either equivalent to a quadrilateral or an annulus. Enriching the topological skeleton with all streamlines that are tangential to feature curves and one streamline for each annulus to cut it into a quadrilateral is sufficient to obtain a valid quad mesh. The necessity of the conditions is also straightforward since violation of (C1) prevents meshing of feature curves, violation of (C2) implies singularities that do not exist in mesh-induced frame field topologies, violation of (C3) generates triangles in the topological skeleton, and violation of (C4) induces a topological skeleton, which is not even a cell-complex anymore. Please note that sufficiency of (C1)-(C4) is only true for 2D domains, while for surfaces embedded in $\mathbb{R}^3$, additional degeneracies exist, e.g. Fig. 5 of [MPZ14].

Local Meshability We call a frame field *locally meshable* if, at each point of the domain, there is an arbitrarily small ϵ-neighborhood that is meshable. Local meshability requires that conditions (C1)-(C3) are fulfilled but omits (C4), which is only relevant on a global scale.



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

Figure 7.4: The polar sector in (a) is modified into the quad sector in (b) by adding an index $\frac{1}{4}$ singularity. Only a local sector neighborhood is altered, depicted with magenta streamlines. Similarly, the anti-quad sector in (c) is modified into a quad sector by adding two singularities of index $\frac{1}{4}$.

Ensuring Local Meshability Given a frame field that satisfies (C1) and (C2), it is possible to turn it into a locally meshable field satisfying (C3), solely by modifying an ϵ-neighborhood of its singularities. As illustrated in Fig. 4ab, a

polar sector at a singularity can be converted into a quad sector by compensating the added rotation with an $index = 1$ singularity. The modification is restricted to an ϵ-neighborhood of the polar sector, and does not alter the field outside. In an analog way, an anti-quad sector can be turned into a quad sector at the cost of locally adding two $index = \frac{1}{4}$ singularities, as depicted in Fig. 7.4cd. While a conceptual understanding of local meshability and modifications to ensure it is sufficient in our context, a concrete algorithm to ensure local meshability in a specific discrete setting by adding sector constraints to the angle-based formulation of [BZK09] can be found in the supplemental material of [MPZ14].

Sector Modifications A generalization of the polar/anti-quad sector removal results in a class of potential *sector modifications*. A modification of a flow-aligned local neighborhood $\mathcal{N}$ can be performed if and only if it respects the index theorem, i.e. $\Delta n_a = \Delta n_q$, with Δn_a, Δn_q quantifying the differences of the total number of anti-quad and quad sectors of all singularities in $\mathcal{N}$. In the example of Fig. 7.4ab, $\Delta n_a = \Delta n_q = 0$ since we add one quad sector at the original singularity but remove one at a regular point, which becomes a singularity of index $\frac{1}{4}$, while the index of the original singularity reduces by $\frac{1}{4}$. For the example of Fig. 7.4cd, $\Delta n_a = \Delta n_q = -1$ since we add one quad sector and remove one anti-quad at the original singularity and remove one quad sector at two previously regular points. Please note that the addition or removal of polar sectors does not require modification of other singularities in the neighborhood, and the same is true for the addition or removal of pairs of quad and anti-quad sectors.

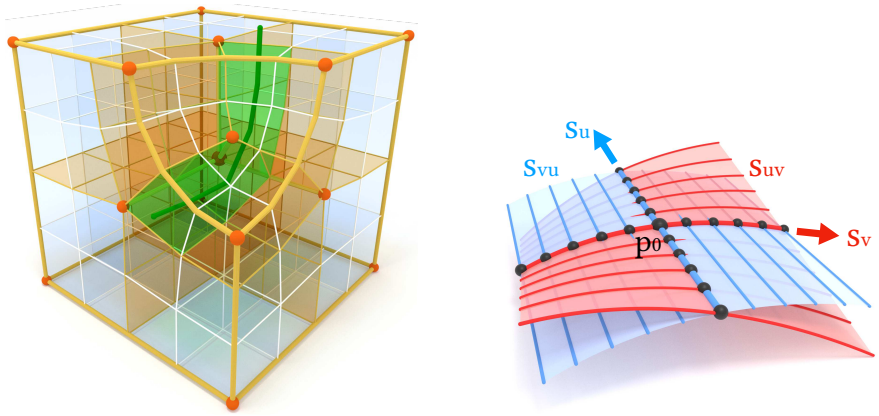
Ensuring Meshability Given a frame field that is locally meshable, it can be turned into a (globally) meshable one with the motorcycle graph based algorithm proposed by Myles et al. in [MPZ14]. The zero loop elimination strategy effectively removes limit cycles, typically at the cost of adding pairs of singularities. We leave the discussion of directly converting the topological skeleton into a meshable one for future work and instead continue with the main topic of this article, frame field meshability in 3D.

7.2 Meshability in 3D

Similarly to the 2D case, a hexahedral mesh in 3D can be seen as a discrete representation of streamlines of a 3D frame field, as illustrated in Fig. 7.5a. The singularities form a graph consisting of *nodes* and *flow-aligned arcs*, referred to as the *singularity graph* [LZC+18]. In contrast to 2D, there are two major differences that complicate the meshabillity of 3D frame fields, i.e., (i) non-uniqueness of streamsurfaces and (ii) singular arcs. Both will be investigated in more detail next.

7.2.1 Streamsurfaces

While under mild continuity conditions, the streamlines of an arbitrary vector field $u(x) : \mathbb{R}^3 \to \mathbb{R}^3$ are uniquely defined by integration, this is no longer true for the streamsurfaces formed by $u(x)$ in combination with a second vector field $v(x) : \mathbb{R}^3 \to \mathbb{R}^3$, as illustrated in Fig. 7.5b. Assume that s_u and s_v are two



(a) (b)

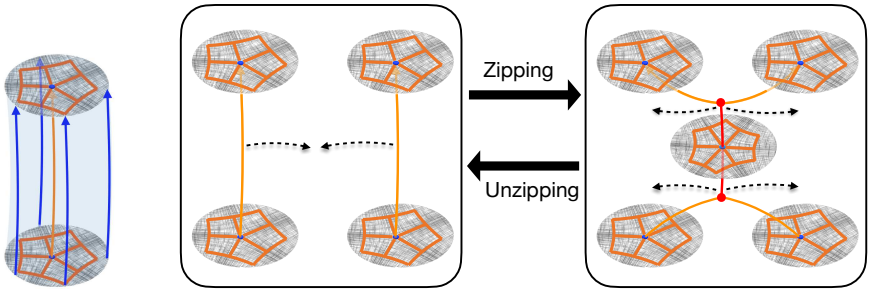
Figure 7.5: (a) a hexahedral mesh induces a frame field with streamlines aligned to all edges and faces of the hexahedral mesh, flowing through (green) channels created by dual chords (dark green) everywhere else with separatrices and separating surfaces in orange. (b) A pair of vector fields u and v is not necessarily integrable. Consequently, the red streamsurface s_{uv} formed by streamlines of v and seeded by s_u passing through p_0 is different from the blue streamsurface s_{vu} generated by exchanging the order of vector fields.

streamlines of u respectively v that intersect at a point p_0. Then s_u serves as a seed curve for a well-defined stream surface s_{uv}, which is formed by all streamlines of v that pass through s_u. However, since the surface resulting from streamlines along v is not necessarily tangential to u, the shape of the surface depends on the order of vector fields in this process. The streamsurface s_{vu} seeded by s_v and formed by streamlines of u, in general, has a different shape, except for the seeds s_u and s_v which by construction are part of both surfaces. Only if the pair of vector fields is *integrable*, meaning that the necessary and sufficient conditions of the Frobenius theorem are satisfied, the streamsurfaces are well defined in the sense that $s_{uv} = s_{vu}$ for any seed point of the domain.

Consequently, unless we can guarantee integrability, e.g., for a hexmesh-induced frame field, there are no well-defined streamsurfaces that partition space into hexahedral cells, preventing a direct extension of the meshability results building on on the topological skeleton in 2D.

7.2.2 Singular Arcs

A local neighborhood of a 2D frame field, called *footprint*, can be extruded along an additional transversal line field to a tubular swept volume equipped with a 3D frame field, as depicted in Fig. 7.6a. If the footprint contains one isolated singularity, a sweep results in a *flow-aligned singular arc*. Such singular arcs are special in the sense that only two vectors of a frame $F = [u, v, w] \in \mathbb{R}^{3\times3}$ vanish, inducing a $rank(F) = 1$ property. The *index* of a flow-aligned singular arc is identical to the index of its footprint. If the footprint contains only quad sectors – a necessary condition for meshability – we call the sweep a (locally) *meshable singular arc*. While hexmesh-induced frame fields contain only meshable singular arcs, general frame fields allow additional non-meshable



(a) (b)

Figure 7.6: (a) A footprint of a 2D singularity is extruded along the blue line field to the orange flow-aligned singular arc. (b) Two orange flow-aligned singular arcs of index $-1/4$ are partially zipped to the red singular arc of index $-1/2$, creating two red zipper nodes.

singular arc types, exhibiting one or several of the following defects:

(D1) *Compound Monodromy* – In contrast to flow-aligned arcs, where the frame field rotates around one axis of the frame, other non-meshable monodromies from the Octahedral group are possible, as explained in more detail below.

(D2) *Not Flow Aligned* – The singular arc is not tangential to the frame field, which prevents meshability and inevitably violates the $rank(F) = 1$ property.

(D3) *Non-constant Footprint* – While the index of a flow-aligned arc is constant, the sectors can still continuously change along the arc (cf. sector modifications of Sec. 7.1.2), e.g. polar sectors opening or closing at separatrices of the footprint.

(D4) *Non-meshable Footprint* – Meshability requires a footprint equipped only with quad sectors, according to (C3).

The *monodromy* of a singular arc specifies the rotation a frame undergoes when traveling along a cycle enclosing the singular arc but no other singularities. When expressed in the coordinate system of the frame itself, the monodromy is an element of the *octahedral group* $\mathcal{O}$, as explained in [CC19]. Only 10 out of 24 octahedral elements correspond to rotations around one axis of the frame and admit meshable singular arcs. According to [JHW+14], we denote the remaining 14 elements as being of *compound type*, which always implies non-meshability of a singular arc, as well as violation of the $rank(F) = 1$ condition, both proven in [NRP11]. Next, we will describe an approach to convert a general frame field into one containing only meshable singular arcs, which is a necessary condition for meshability of a frame field.

Arc Zipping A fundamental operation for modifying singular arcs and singular nodes are *zipping* and *unzipping*, both depicted in Fig. 7.6b. Given two parallel flow-aligned singular arcs A_1 and A_2 with index I_1 and I_2 respectively, the process of letting the arcs approach each other until they partially merge at novel flow-aligned arc A_z is called *zipping*. The index of A_z is $I_z = I_1 + I_2$. The inverse process is called *unzipping*, and the resulting indices can be any

I_1 and I_2 satisfying $I_1 + I_2 = I_z$. Like a zipper that can open and close continuously, both operations can be restricted to sub-segments of arcs, in which case singular nodes, called *zipper nodes*, are created at the end points of the (un)zipping operation, or entirely merge two arcs. These operations are the 3D analog of the sector modifications of Sec. 7.1.2, acting inside the swept volume of a flow-aligned singular arc. A regular streamline, i.e. a flow aligned arc of zero index, can be unzipped into arcs of opposite sign $I_1 = -I_2$. Such a pair of index $\pm\frac{1}{4}$ arcs serve as generators for all other meshable arcs under the zipping operation since they correspond to the addition and removal of one quad sector in the footprint, respectively. Please note that arc zipping operations might be restricted by feature constraints, where streamlines are not allowed to deform continuously in arbitrary directions. One frequent special case is a zipper node that reaches the boundary and belongs to a transversal flow direction, in which case the zipper node can be dissolved by disconnecting the two arcs.

Ensuring Meshable Singular Arcs A general frame field can be converted into one with solely meshable singular arcs by successively removing defects of types (D1)-(D4). As observed in [JHW+14], all non-meshable monodromies are the product of two meshable monodromies. Consequently, a singular arc of compound type can always be unzipped into two parallel arcs of non-compound type, which either form two parallel closed orbits or connect at common singular nodes. Modification of the frame field is only required in an arbitrarily small local neighborhood of the compound arc.

Defects of type (D2) of a non-compound arc can be resolved by a continuous transformation of the frame field in an epsilon neighborhood such that the field becomes tangential to the arc. The goal is to align the frame field while minimizing the required change, e.g., a Dirichlet type energy subject to alignment constraints deduced from the angles of the non-aligned field. Please note that the monodromy of a non-aligned arc specifies the tangential axis, as well as the index, only up to its sign. If the choice of alignment axis changes along the arc, singular nodes need to be introduced, which split the arc into several segments of index with alternating signs.

Defects of type (D3) are resolved by splitting an arc into sub-arcs of constant footprint by introducing singular nodes at the transition points.

All defects of type (D4) can be repaired by zipping. For instance, the repair of a polar sector in a footprint is done by unzipping an index $\frac{1}{4}$ singular arc from the polar sector, similarly to Fig. 7.4ab.

After ensuring that all singular arcs are of meshable type, all remaining defects preventing local meshability can only be located at singular nodes, which will be discussed next.

7.2.3 Singular Nodes

In a hexmesh-induced frame field all *singular nodes* – branching points where several singular arcs meet – are isolated point singularities with $||F||_2 = 0$, a condition identical to (C2) in 2D. All singular node types existing in hexmesh-induced frame fields are called *meshable* [LZC+18]. First, we will clarify that singular nodes always result from the interaction of singular arcs and that me-

shable arcs of index $\pm\frac{1}{4}$ not only serve as generators for all meshable arcs but also as generators for all meshable nodes.

Theorem 1. *All types of meshable singular nodes can be constructed by iteratively zipping meshable singular arcs of index* $\pm\frac{1}{4}$.

Proof. The intersection of the one-ring hex mesh of a meshable singular node with the surrounding 2-sphere induces a triangulation of the 2-sphere, as observed in [NRP11]. Each edge of the triangulation corresponds to one quad sector, belonging to the two flow-aligned arcs represented by vertices opposite to the edge. From this perspective, zipping a meshable singular arc of index $\frac{1}{4}$ corresponds to an edge collapse (removing a quad sector), while zipping one of index $-\frac{1}{4}$ is equivalent to an edge split (adding a quad sector). All potential triangle meshes of genus zero can be generated starting from an octahedron (regular node) by a sequence of edge collapsing and splitting. Consequently, all meshable singular nodes can be generated by iteratively zipping meshable singular arcs of index $\pm\frac{1}{4}$. □

Zipper Nodes Not all singular nodes resulting from the interaction of meshable singular arcs are meshable. While meshability of arcs ensures that their footprint is intact, it is still possible that the two of them converge in a parallel fashion to a singular node, creating a (non-meshable) parabolic sector in the vector field on the branched cover. Since such a singular node, where two arcs meet in a non-meshable fashion, always results from an incomplete zipping of two singular arcs, we call them *zipper nodes*. Unfortunately, zipper nodes are not rare and can frequently be observed in frame fields that are optimized for smoothness, for instance, for the notch model shown in Fig. 7.7a. The feature constraints on top require two singular arcs of index $\pm\frac{1}{4}$. However, the quadrilateral base favors a constant frame field. Consequently, the smoothest frame field generates a zipper node, where the two singular arcs of opposite index meet in a parabolic sector and then continue as a regular arc.

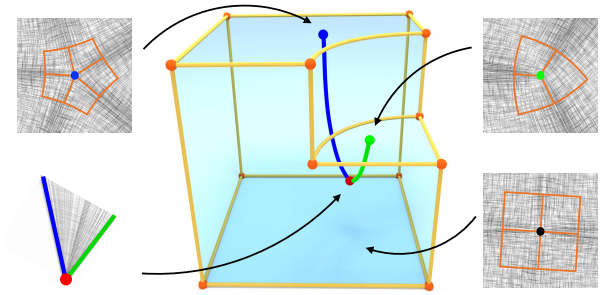
Repairing Fundamental Zipper Nodes There are three different strategies to get rid of *fundamental zipper nodes* that are formed by a pair of singular arcs with opposite index, illustrated in Fig. 7.7bcd.

(S1) *Zipping – Strategy I consists of getting rid of both singular arcs by entirely zipping them together to a regular streamline..*

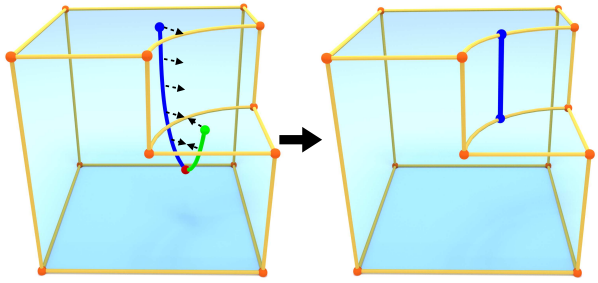
(S2) *Unzipping* – Strategy II consists in unzipping along the regular arc that is the continuation of the two singular arcs terminating at the zipper node.

(S3) *Polar Sector Repair* – Strategy III consists in leaving the arcs meeting at the zipper node untouched but instead unzipping the regular streamline that passes through the tip of the parabolic sector, generating a pair of $\pm\frac{1}{4}$ singular arcs. The $-\frac{1}{4}$ arc turns the parabolic sector into a quad sector, as depicted in Fig. 7.7d.

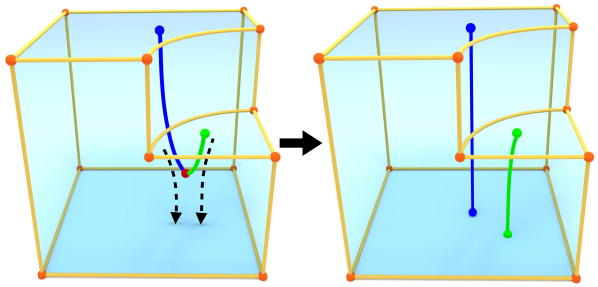
Strategies S1 and S2 can be impossible due to constraints of feature curves or feature surfaces that prevent the operation. However, S3 is always valid. Please note that for strategies S2 and S3, the frame field could be slightly modified



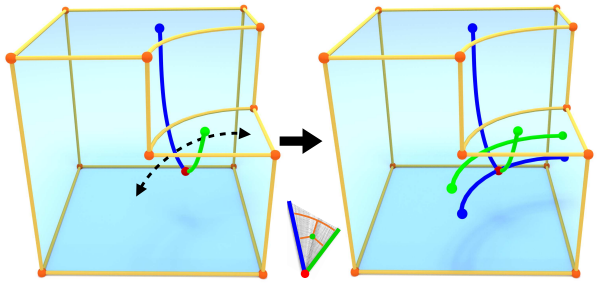
(a)



(b)



(c)



(d)

Figure 7.7: (a) the smoothest frame field on the notch model contains a zipper node since the features on top require singularities, while the base favors a constant frame field without singularities. (b) resolving a zipper node by zipping, (c) resolving a zipper node by unzipping, (d) resolving a zipper node by unzipping a regular streamline and attaching the $-\frac{1}{4}$ arc to the parabolic sector at the zipper node, turning it into a quad sector.

before unzipping along the regular streamline to ensure that the boundary of the domain is reached without getting caught in potential limit cycles. Identical strategies have been discussed in [RCR19], where fundamental zipper nodes are called non-meshable 3-5 arcs.

Repairing Non-Meshable Nodes Fundamental Zipper nodes are only one particular type of non-meshable nodes, and we need a general strategy to repair other types of non-meshable nodes. Assuming that all singular arcs are of meshable type, it is possible to fully decompose singular nodes such that only valid singular arcs and fundamental zipper nodes remain.

Theorem 2. *Every singular node of a frame field that is only incident to meshable singular arcs can be decomposed into a finite set of isolated meshable singular arcs and isolated fundamental zipper nodes by only modifying an arbitrarily small local neighborhood.*

Proof. It suffices to observe that each quad sector of a singular arc A_1 continues along at least one other singular arc or regular streamline A_2 incident at a shared singular node. If A_1 and A_2 are both singular, unzipping an arc with index $I_d = \max(I_1, I_2)$ will detach a singularity with index I_d from the node. The corresponding arc will become a regular streamline, or even both, if $I_1 = I_2$. This process can be repeated until either all singular arcs have been detached from the node, or a set of pairs of singular arcs forming parabolic sectors remains. In the latter case, the pairs can be iteratively detached by zipping, i.e. generating an isolated fundamental zipper node that connects to the original singular node with a regular streamline. □

Theorem 2 is a generalization of the decomposition of singularities of a hexahedral mesh of the concurrent work [PJHHXCK22] to the setting of frame fields with meshable singular arcs but potentially non-meshable singular nodes.

After the decomposition of all non-meshable singular nodes, the only remaining non-meshable configurations are fundamental zipper nodes. After resolving all of them by (S1), (S2), or (S3), all singular arcs and nodes are meshable. Consequently, the result is guaranteed to be a locally meshable frame field.

Next, we will refine the conceptual algorithm presented in this section to a practical one for piecewise constant frame fields on tetrahedral domains.

7.3 Local Non-meshabilities and Repairs

Given an input tet mesh $\mathcal{T} = (V, E, T, C)$ with vertices V, edges E, triangles T, cells C, and a smooth frame field $\mathcal{F}$ from which the singularity graph $\mathcal{S}$ embedded in the 1-skeleton $V \cup E$ is extracted, the goal is to ensure that $\mathcal{F}$ in one-ring vicinity of $v_i \in \mathcal{V}$ is locally meshable. We consider octahedral frames in this section and represent them with unit quaternions q. Each tetrahedron defines a chart and holds a local coordinate system. The connection between charts c_i and c_j is established by matching $R_{i\to j} \in \mathrm{Oct} : q_j = R_{i\to j} q_i$, where Oct is the octahedral group. The discretized octahedral frame field $\mathcal{O} = (\mathcal{R}, \mathcal{Q})$ can be expressed as a set of matchings $\mathcal{R}$ belonging to oriented dual edges $e^\star$, and

a set of unit quaternions $\mathcal{Q}$. The tet mesh explicitly encodes feature properties, including feature vertices V_F, edges E_F, and faces F_F, on the boundary or in the interior of the domain, which imposes alignment constraints to the frame field. This section identifies local non-meshable defects in tet meshes with feature tags and proposes repair operations for the practical scenario which are analogous to the conceptual algorithm discussed in Sec.7.2. We begin with the core operations *arc zipping* and *unzipping*, which locally modify frame field topology.

7.3.1 Arc Zipping

Assume a disk $\mathcal{D}$ which consists of a series of triangles that is bounded by oriented singular arcs A_1 and A_2, as shown in Fig.7.8, and is free of other singularities. Zipping the arc A_2 to A_1 would change the index I_1 of A_1 to $I_1 + I_2$ and I_2 of A_2 to *identity*. Modifying the index of an arc requires the adjustment of topological matchings on $\mathcal{D}$. Consider the one-ring tet mesh of an oriented edge e. The rotation specified by the edge monodromy M_e is measured via the concatenation of matchings between neighboring frames: $R_e = R_{k\to 0}...R_{1\to 2}R_{0\to 1}$ w.r.t. to the local coordinate system in chart c_0. R_e is denoted as $\mathrm{rot}(u, \alpha)$, where $u \in (\pm x, \pm y, \pm z)$ is the one of the frame axes in c_0, and α corresponds to the index of e with $\alpha = 2\pi I_e$. The matching $R_{i\to i+1}$ to modify on the incident half-face of e on $\mathcal{D}$ is thus determined via:

$$R_{i\to i+1} = (R_{k\to 0}...R_{i+1\to i+2})^{-1}\mathrm{rot}(u, \alpha)(R_{i-1\to i}...R_{0\to 1})^{-1}.$$

Knowing the target indices of A_1 and A_2 after *zipping* and their rotation axes w.r.t. chart c_0, we can determine matchings on half-faces incident to A_1 and A_2 on $\mathcal{D}$. The rest matchings on $\mathcal{D}$ can be solved by iteratively applying the *chart zippering* [LZC+18]. Note that since the matching is invariant to rotations of multiple 2π, zipping an arc A_2 of a higher index to A_1 is not possible via directly modifying the matching on $\mathcal{D}$. However, with the matching modification, zipping the singular arcs of index $\pm\frac{1}{4}$ is unique. Provided enough DOFs, singular arcs of an arbitrary index can be generated by successively zipping singular arcs of index $\pm\frac{1}{4}$ through different $\mathcal{D}$. As the inverse process of *zipping*, *unzipping* can be realized similarly via modifying the matchings on $\mathcal{D}$ such that the target indices of A_1 and A_2 are fulfilled.

Zipping and *unzipping* singular arcs of index $\pm\frac{1}{4}$ are the fundamental operations upon which all repair operations are build. They differ in the construction of the disk $\mathcal{D}$ and the determination of $\mathrm{rot}(u, \alpha)$ such that corresponding local defects are fixed. Next, we will classify different types of non-meshable local cases in tet meshes considering that feature constraints are involved, and introduce corresponding operations to ensure local meshability.

7.3.2 Ensuring Meshable Edges

Repair Compound Singular Edge The monodromy $M_{e_1} = \mathrm{rot}(u, \alpha)$ of a compound singular edge e_1 specifies a 3D rotation, resulting in rotational symmetry of the cube, whose rotational axis is not from the center of a face to the center of the opposite face. Such rotations are guaranteed to be decomposed into two rotations along principle axes. We take an incident triangle of e_1 as the disk $\mathcal{D}$ which contains no other singular edges and then search for a meshable

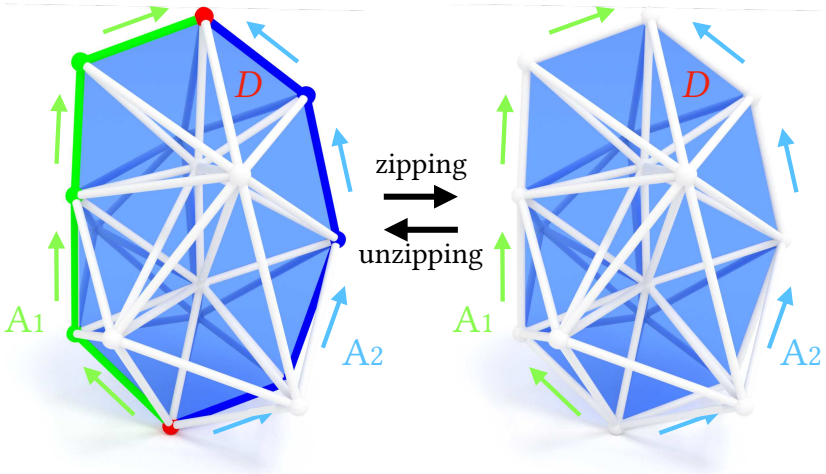
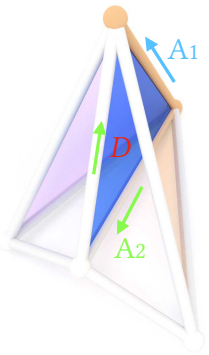


Figure 7.8: Zipping and unzipping a pair of arcs A_1 and A_2. Green/blue indicates index $\frac{1}{4}/-\frac{1}{4}$.

monodromy M_{e_2} of a regular edge e_2 on the triangle such that M_{e_1} is converted to a meshable monodromy after unzipping. This is similar to the splitting approach in [JHW+14]. Another way to repair compound singular edges is by removing them with the edge collapsing [LLX+12] operation.

Repair Polar Sector Wedge The polar sector wedge corresponds to local defects of types (D4) with feature faces (in purple/orange) incident at feature arc (in orange) A_1 forming a wedge of polar sectors. Each polar sector wedge is repaired via unzipping a singular arc of index $\frac{1}{4}$ from it into the wedge. Splitting guarantees the construction of a disk $\mathcal{D}$ inside the wedge such that it does not enclose any singularity. When expressed in the same chart, the axis u of arc A_2 is identical to the aligned frame axis of A_1, and the angle α is $\frac{\pi}{2}$. The rotation of the target monodromy of A_2 for the unzipping is thus $\mathrm{rot}(u, \frac{\pi}{2})$. Note that when all polar sector wedges are converted to quad sectors, the defects of type (D3) will be automatically fixed since the footprint cannot change along singular arcs.



Locally Aligning and Smoothing Field According to the condition (C1) and defect (D2), a meshable frame field has to fulfill alignment constraints, categorized as face $\mathcal{C}_f$ and edge alignment constraints $\mathcal{C}_e$. Precisely, one of the principle axes $(\pm x, \pm y \pm z)$ of the frame q_i in a cell $c_i \in C$ should be aligned to the normal direction of the incident feature triangle, and the singular/feature edge direction. If multiple constraints of the same type exist in a single cell, either *face split* or *edge split* is necessary to ensure constraints free of conflicts. Frame field optimization with fixed matchings and alignment constraints can be formulated as in [LZC+18]. Instead of relaxing it to an eigenvalue problem, we iteratively optimize each quaternion q_i by averaging its neighboring quaternions q_j and then project it to the constrained axis. Consider a set of connected tets

$\mathcal{T}$ with intial octahedral field $\mathcal{O} = (\mathcal{R}, \mathcal{F})$ and alignment constraints $\mathcal{C}$. The optimization procedure is described in Algo. 5.

Algorithm 5 LocallyAlignedSmoothField

Input: Tet mesh $\mathcal{T}$, initial field $\mathcal{O} = (\mathcal{R}, \mathcal{Q})$ and constraints $\mathcal{C}$
Output: Optimized field $\hat{\mathcal{O}} = (\mathcal{R}, \hat{\mathcal{Q}})$

```
1: while i ≤ NumIters do                                   ▷
2:     for c_i in T do
3:         for c_j in N(c_i) do          ▷ N(c_i): neighbouring tets of c_i
4:             q_i ← q_i + R_{j→i} q_j                     ▷
5:         end for
6:         project(q_i)                                    ▷
7:         normalize(q_i)                                  ▷
8:     end for
9: end while
```

Projection. The optimization strategy decouples smoothing and aligning steps by applying a projection after averaging neighboring quaternions. Owing to splitting, q_i in a cell can have at most one constraint of type C_f and C_e each. When their aligned axes in chart c_i are linearly independent, the constraints fully determine the frame. Otherwise, one constraint only specifies one axis that a frame q needs to align. Assuming the partially aligned axis x and the direction $d \in \mathbb{R}^3$, the rotation R brings q to the coordinate system where d is the direction of x axis: $\hat{q} = Rq$. The projected quaternion is thus $\hat{q}_p = (q.w(), q.x(), 0, 0)$. Normalizing it and then expressing it back in the original coordinate gives $q_p = R^{-1}\hat{q}_p$ [GJTP17].

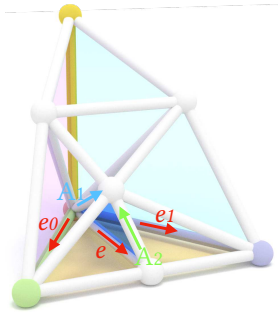
Double Covering. Since quaternions q and $-q$ represent the same rotation, the sign of matchings in the quaternion form should be determined during the averaging. We measure the dot product of q_i and $R_{j\to i}q_j$ and revert the sign of $R_{j\to i}q_j$ if the dot product is negative.

7.3.3 Ensuring Meshable Vertices

Provided meshable edges at a vertex, Sec.7.2.3 presents a conceptual algorithm to decompose a general singular node into meshable singular arcs and zipper nodes. Then three strategies are proposed to repair the remaining zipper nodes. However, the existence of features in tet meshes complicates the repairing problem as features impose topological constraints on frame fields. Besides polar sectors formed by two singular edges (singular-singular), they can also be formed between a feature edge and a feature edge/face (feature-feature), and a singular edge and a feature edge/face (singular-feature). For repairing a non-meshable vertex on the feature, we propose first repairing all incident feature-feature polar sectors and then applying the decomposition algorithm while maintaining the meshability of feature-feature sectors. Cases directly detaching singular arcs from the vertex induce feature-feature or feature-singular polar sectors are classified as constrained polar sectors. We describe in more detail below the necessary steps for ensuring meshable vertices, including: (i) feature-feature polar sector repair, (ii) singular node decomposition, (iii) constrained polar sector repair, and (iv) zipper node repair through a series of basic operations of

zipping/unzipping. While repairing non-meshable vertices, the meshability of edges is preserved due to splitting.

Feature-feature Polar Sector Repair We assume that each feature edge is incident to at least one feature face as in [BRK+22]. Due to alignment constraints, the frame field in the one-ring mesh of the feature vertex is partitioned into sectors in the supporting plane by incident feature edges. The defect of the feature-feature polar sector corresponds to the type of non-meshable footprint, which can be repaired by unzipping a singular arc of index $\frac{1}{4}$ into the sector. Consider the polar sector (yellow) with bounding feature edges e_0, e_1 and a regular edge e in the sector. In the unzipping operation, we specify the disk $\mathcal{D}$ (blue) as the non-feature triangle incident to e, with its third vertex not on any feature patch. In determining the target monodromy of arc A_1, the rotation axis u is the aligned frame axis to the normal direction of the polar sector and α is $-\frac{\pi}{2}$ since the target index of A_1 is $-\frac{1}{4}$. The unzipping opens the polar sector to a quad sector at the cost of introducing a zipper node. Note that the repair operation should be applied twice for interior feature-feature sectors since e_0 and e_1 bound two oriented sectors with opposite normals.

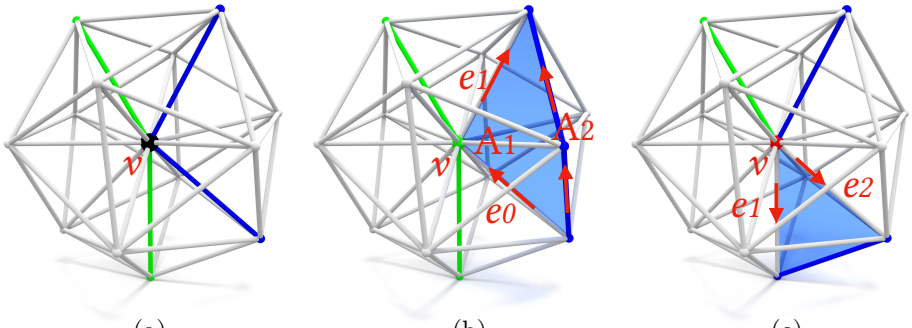


The feature-feature polar sector repair can be generalized to the case where isolated feature curves exist. The extra effort is to construct a supporting plane between two feature edges, or a feature edge and a feature face, and approximate the aligned frame axis to the normal of the plane.

In the smooth frame field, the other type of non-meshable footprint, the elliptic sector, is not observed. Nevertheless, repairing such sectors is possible via the approach described in 7.1.2.

Singular Node Decomposition Consider a non-meshable singular node v with all incident edges meshable. Following the conceptual algorithm in Theorem 2, we repair the invalid singular node by iteratively detaching meshable singular arcs of index $\pm\frac{1}{4}$ via arc unzipping and zipper nodes via zipping from it until it becomes locally meshable or a zipper node.

Detaching singular arcs. In the one-ring mesh at v, given a start singular edge e_0 with $Me_0 = \text{rot}(u_0, \alpha)$ in the incident chart c_0, we search for the other parallel singular edge e_1 which forms the singular arc A_1 with e_0 and then perform the standard unzipping such that A_1 becomes regular. e_1 is parallel in the sense that the rotation axis of e_1 expressed in c_0 through a dual path $\mathcal{P}$ is identical to u_0. As the monodromy is known, the missing piece for the unzipping is the disk $\mathcal{D}$ (blue), which is simultaneously constructed while seeking e_1. Specifically, we perform a breadth-first search for the path $\mathcal{P}$ of dual edges with matching product $R_{\mathcal{P}} = R_{k-1\to k}...R_{1\to 2}R_{0\to 1}$, which reaches a cell c_k at a singular edge e_1 with monodromy $R_{\mathcal{P}}\text{rot}(u_0, \alpha)$. The search starts from c_0 in the direction of $\hat{u}_{e_0}$ which is the edge axis of e_0 in chart c_0: $\hat{u}_{e_0} = u_0$ if $\mathbf{e}_0 \cdot F_0(u_0) \geq 0$, otherwise $-u_0$. We confine the search space in each chart to a cone region to avoid rotations of multiple 2π, i.e. while propagating from chart

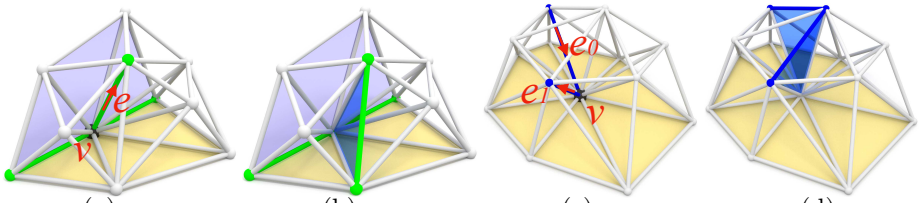


(a) (b) (c)

Figure 7.9: An example of repairing a singular node. (a) Two parallel singular arcs touch, resulting in an invalid singular node (black). (b) Detaching the blue arc from v repairs the non-meshable singular node. (c) The other way to decompose v by detaching a zipper node, with v turned into a zipper node. In this section, singular edges of indices $\frac{1}{4}/-\frac{1}{4}$ are in green/blue, regular edges in white, and feature edges in other colors.

c_i to c_{i+1}, the angle β defining the cone between the normal n_i of the face $f_{i\to j}$ and the search direction $F_i(\hat{u}_i)$ is less than 100°. Edges e_0 and e_1 separate the bounding surface of $\mathcal{P}$ that is in the interior of the one-ring mesh into two disks, either of which can be used as the disk $\mathcal{D}$ for the unzipping. Fig.7.9b depicts the detaching process of a singular arc A_1 of index $-\frac{1}{4}$.

Detaching zipper nodes. A zipper node is detached via zipping a given singular edge (arc) e_1 (A_1) with $Me_1 = \text{rot}(u_1, \alpha)$ and the other target singular edge (arc) e_2 (A_2) with $Me_2 = \text{rot}(-u_1, -\alpha)$ w.r.t. a common chart c_1. Here, e_1 and e_2 are both outgoing edges at v. The searching for e_2 and the construction of the disk $\mathcal{D}$ are similar to the process in *detaching singular arcs* except that the searching angle $\beta \geq 45°$. Often detaching a zipper node reduces the number of polar sectors at v but introduces a new zipper node in the vicinity (c.f. Fig.7.9c). To minimize the number of zipper nodes, detaching singular arcs takes priority over detaching zipper nodes in the decomposition.



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

Figure 7.10: Feature-singular polar sector repair with singular node decomposition. Feature surfaces are colored in purple and yellow with transparency. (a) The edge e ends at a boundary vertex v and it is orthogonal/tangential to the yellow/purple surface, respectively. (b) Detaching e from v repairs the polar sector. (c) Parallel edges e_0 and e_1 tangentially touches the feature surface at v. (d) Detaching the singular arc from v makes it meshable.

Besides the singular-singular polar sectors, the singular node decomposition can also be used to repair feature-singular polar sectors, i.e., a singular edge e_0 off feature surface tangentially touching the feature, e.g., Fig.7.10(c). Such cases can be repaired by detaching the singular arc/zipper node if the other

corresponding singular edge e_1 can be found, as shown in Fig.7.10(d). A special case is that e_0 orthogonally ends on the boundary surface, e.g., Fig.7.10(a), and the other counterpart e_1 does not exist. The disk path $\mathcal{D}$ is constructed via a dual path $\mathcal{P}$ which connects e_0 and a boundary face whose normal is parallel to the edge axis of e_0, see Fig.7.10(b).

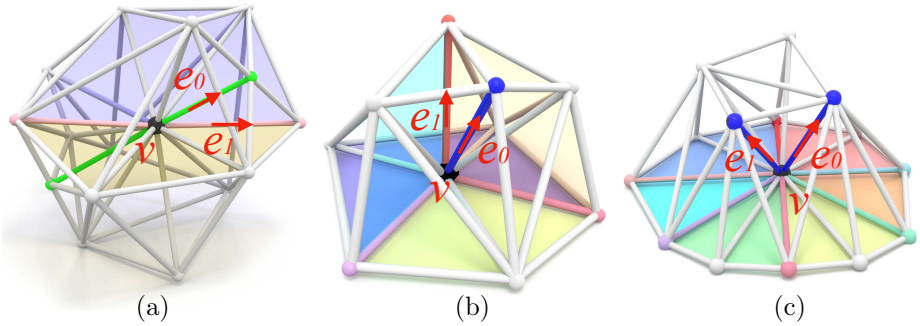
There are two conditions for detaching singular arcs/zipper nodes at a vertex on feature: (i) the dual path $\mathcal{P}$ should orthogonally cross the feature surface or end on the boundary, ensuring that no new feature-singular polar sectors will be generated, and (ii) the operation should not induce any feature-feature polar sector. Consequently, the singularity decomposition cannot be applied in the category which we refer to as the constrained polar sector. A different zipping strategy for repairing constrained polar sectors will be discussed in the next section.

Constrained Polar Sector Repair A constrained polar sector is a feature-singular or singular-singular polar sector formed by an oriented singular edge e_0 of with its from vertex v on feature edges and the other entity (feature edge/face or singular edge). Depending on the local configuration, there are three basic types of constrained polar sectors:

1. A singular arc crosses the interior feature surface at a feature edge vertex v and the edge axis $\hat{u}_{e_0}$ is tangential to all feature patches (c.f. Fig.7.11(a)). e_0 forms a polar sector with the feature edge e_1.

2. A singular edge e_0 of index $-\frac{1}{4}$ forms a polar sector with a feature edge/-face at the vertex v, where all feature-feature sectors whose normals are parallel to the edge axis $\hat{u}_{e_0}$ are quad sectors. Fig.7.11(b) shows a special case where a feature edge e_1 forms a polar sector with e_0.

3. A singular edge e_0 of index $-\frac{1}{4}$ forms a polar sector with another parallel singular edge e_1 with the same index, and all orthogonal feature-feature sectors are quad sectors as in (2), c.f. Fig.7.11(c).



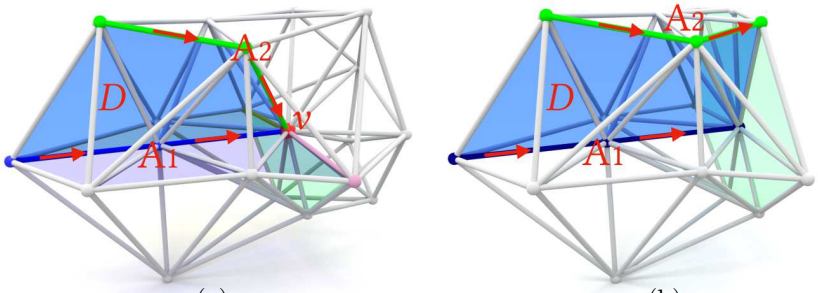
(a) (b) (c)

Figure 7.11: Three types of constrained polar sectors. (a) The singular arc tangentially passes through the feature surfaces at v. (b) The singular edge e_0 of index $-\frac{1}{4}$ is parallel to the feature edge e_1 and the orthogonal feature-feature sector (yellow) in the same region is a quad sector. (c) Two singular edge e_0, e_1 of index $-\frac{1}{4}$ are parallel and all orthogonal feature-feature sectors are quad sectors.

In case (1), directly detaching the singular arc would result in a new feature-singular polar sector, as there is no disk path that orthogonally crosses the feature surface. For types (2) and (3), we may detach e_0 from v by unzipping a singular arc of index $-\frac{1}{4}$ into a feature-feature sector, however, at the cost of shrinking it to a feature-feature polar sector which is illegal. An intuitive repair would be to zip the singular arc A_0 (containing e_0) to the singular/feature arc A_1 (containing e_1). But there is no guarantee of constructing a disk $\mathcal{D}$ bounded by A_0, A_1 with no other singular arcs passing through.

One way to repair the constrained polar sector is by unzipping the arc A_1 of index I_1 into the arc A_1 of index $I_1 - \frac{1}{4}$ and a new arc A_2 of index $\frac{1}{4}$. Owing to splitting, the unzipping is ensured in the one-ring vicinity of the arc A_1, and a disk $\mathcal{D}$ free of other singularities is guaranteed. Afterwards, A_2 of index $\frac{1}{4}$ and e_0 of index $I_1 - \frac{1}{4}$ form a zipper node at v which can be repaired by detaching zipper node. In the process, we require that the repair operation would not introduce new constrained polar sectors. The key is about searching for the guiding arc A_1.

For types (2) and (3), the search for A_1 is in the same direction as the edge axis $\hat{u}_{e_0}$, expressed in an incident chart of e_0. During the marching, A_1 follows the feature or singular edge, if any, which is parallel to the marching direction. Otherwise, we continue the propagation in the search direction either on the feature surface for type (2) or inside the volume for type (3). A_1 either stops at an orthogonal feature patch in the interior of the mesh or ends orthogonally at the boundary of the mesh. In the former case, unzipping A_1 results in a zipper node at the end vertex v_e (c.f. Fig.7.12) while in the latter case, v_e is meshable.



(a) (b)

Figure 7.12: (a) A_1 ends at an orthogonal feature surface (green) and introduce a zipper node at v_e. (b) A_1 orthogonally ends at the boundary surface (green).

For type (1), the marching direction can be either along e_1 or the opposite feature edge. The dihedral angle between feature faces in frame space at one of the two feature edges at v must be π but not the other. We pick the feature edge with the sector angle unequal to π to begin for consistency. Depending on the search direction, the index of A_2 after unzipping is either $-\frac{1}{4}$ or $\frac{1}{4}$. Therefore, in the former case, it may result in another constrained polar sector of type (2) at the end vertex v_e, which can be fixed subsequently. The stopping condition for searching A_1 is the same as type (2) and (3). Furthermore, the marching can terminate when it reaches a singular edge e_s with the same rotation axis as e_0, expressed in the same chart. e_s should locate in the same region as A_2 so that a singular arc detaching operation can be executed between A_2 and e_s. Overall, constrained polar sector repair operation does not produce any other

type of defect except for the zipper node.

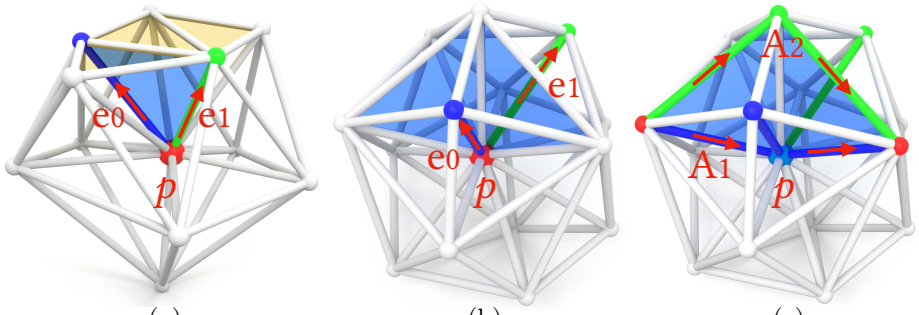
Zipper Node Repair The above repair operations would make a vertex locally meshable or convert it to a zipper node. Sec.7.2 conceptually proposes three strategies for repairing zipper nodes. We discuss equivalent strategies of repairing the zipper node of frame fields embedded in tet meshes, i.e., handling a zipper node p with outgoing singular edges e_0 and e_1 by means of *zipping*, *unzipping* and *polar sector repair*.

Zipping. Strategy I zips the outgoing singular arcs A_1 and A_2 at the zipper node p such that the two arcs cancel out each other. Particularly, knowing the target monodromies of A_1 and A_2, a disk $\mathcal{D}$ spanned by the singular arcs, not enclosing any other singularities, is required for the zipping. Constructing such a disk for general singular arc pairs is challenging, sometimes even impossible, but is trivial in the one-ring neighborhood. For example, when e_0 and e_1 touch the boundary, as shown in Fig.7.13(a), we choose the zipping operation to remove the zipper node.

Unzipping. Strategy II unzips a regular streamline into a pair of singular arcs of opposite indices $\pm\frac{1}{4}$. Exactly tracing the streamline from p is non-trivial since the frame field is often not integrable and thus might get stuck in a limit cycle. Instead, an approximation is feasible since rotational axes of e_0 and e_1 are well defined in the parametric domain, indicating the search direction of the regular arc. It either ends at the boundary surface with the normal axis agreed to the search axis $\hat{u}$ or at another zipper node p' where its searching axis $\hat{u}'$ is in the opposite direction. An illustration of *unzipping* is in Fig.7.16(c)-(d). Specifically, taking an incident chart c_0 of e_0 as a reference coordinate system and p as the origin, we search for a dual path $\mathcal{P}$ where the regular arc is embedded. The propagation starts from c_0 in the direction of $\hat{u}_0 = -\hat{u}_{e0}$ and stays inside the cone defined as $d = u^2 + s(v^2 + w^2), u > 0$, where u, v, w are the coordinates in parametric space and s controls the shape of the cone. $\mathcal{P}$ is closer to the iso-line (u, v_p, w_p) with a larger s. Instead of asking for a strictly increasing $\mathcal{P}$ in the u coordinate, we forbid $\mathcal{P}$ to proceed if the angle between the outward face normal n_{ij} and the search direction $\hat{u}_i$ in chart c_i is below $100°$.

The dual path $\mathcal{P}$, which is a strip of face-connected tets, ends at the boundary or another zipper node p'. We search for two non-intersected arcs A_1 and A_2 on the surface of $\mathcal{P}$ starting from the endpoints of e_0 to the boundary or another zipper node p'. The first edge path A_1 is grown w.r.t. a difference vector d_v in chart c_0, attempting to maximize the projection of the edge path on d_v, which is the edge difference $e_0 - e_1$ expressed in chart c_0 and projected on (v, w) plane. The second edge path A_2 can be found via *Dijkstra* without touching A_1. They separate the surface of $\mathcal{P}$ into two disks, either of which can be the disk $\mathcal{D}$ for unzipping. Unzipping would turn e_0 into a regular edge and A_1, A_2 into a pair of meshable singular arcs of opposite indices.

Polar sector repair. Strategy III is implemented similarly to the repair of feature-feature polar sectors. The difference is that the sector spanned by e_0 and e_1 is not well-defined in parametric space since it degenerates into a segment. However, in smooth frame fields that are not singularity-aligned during the repairing stage, the sector normal can be approximated in a larger neighborhood by integrating along the outgoing singular arcs of p. The normal of the



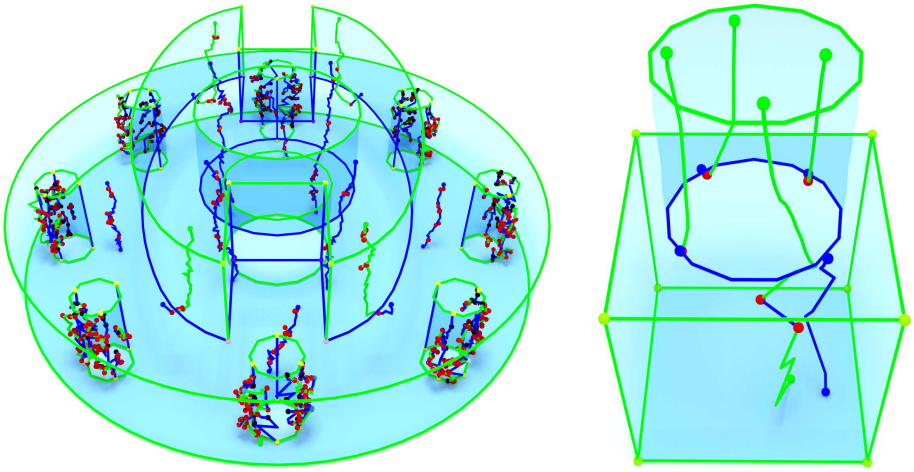
(a) (b) (c)

Figure 7.13: Zipper node repair. (a) Zipping the singular edge pair which touches the boundary surface (yellow). (b) A disk (blue) intersected with the sector formed by e_0 and e_1. (c) Opening the sector by polar sector repair converts the zipper node to a meshable singular node of signature (1,3,3) [LZC+18].

approximated plane indicates the rotational axis u for unzipping. Fig.7.13(b)-(c) show that a zipper node is converted to a valid singular node of signature $(1, 3, 3)$[LZC+18] by unzipping into singular arcs A_1 and A_2. It creates two additional zipper nodes in the one-ring vicinity. In practice, if a regular edge connects p to another arc of index $-\frac{1}{4}$, which orthogonally passes through the sector at p, we simply collapse the regular edge to open the polar sector.

Unzipping is the primary operation for repairing zipper nodes since zipping requires a challenging construction of disk $\mathcal{D}$ without enclosing any other singularities, and polar sector repair introduces two additional zipper nodes that need further treatment. In the rare case when singular arcs of a zipper node end at the boundary patch of a limit cycle [VSL16], unzipping would fail to find a valid dual path. We can remove it by an iterative zipping or applying the polar sector repair which transforms it into two zipper nodes of the type that unzipping can repair. Practically, unzipping and zipping are sufficient to repair zipper nodes, while polar sector repair serves as a complementary option.

Algorithm. With the above operations, an intuitive pipeline to repair a general frame field follows the steps: (1) ensuring meshable edges; (2) ensuring meshable vertices; (3) aligning the frame field to singular or feature edges. Ideally, this approach would ensure that every tet mesh vertex is locally meshable. However, in practice, the algorithm often either does not produce locally meshable frame fields or it does generate locally meshabe frame fields, but with high distortions that are not practically valuable. The singularity graphs of input frame fields are geometrically noisy, containing many redundant zipper nodes, as shown in Fig.7.14a. Simply re-labeling the index of singular arcs and re-aligning the frame field to remove redundant zipper nodes would induce significant distortion. Moreover, the repair operations in tet meshes of finite resolution often result in zigzag singular arcs, bringing in additional zipper nodes (c.f. Fig.7.14b). As a result, a large number of zipper nodes may need to be repaired, leading to over-complex singularity structures. In section 7.4, we propose a practical algorithm that combines continuous frame field optimization, singularity relocation, and singularity repairing to produce locally meshable frame fields.



(a) (b)

Figure 7.14: (a) Singularity graph of the input frame field of *i01c* contains many redundant zipper nodes (red). (b) In *s11c_cube_cyl*, unzipping a zipper node induces two redundant zipper nodes on the pair of singular arcs due to geometric noise.

7.4 Practical Algorithm

Given a feature-aligned smooth frame field, we aim to generate locally meshable frame fields with reasonable singularity complexity. With this in mind, we devise a practical algorithm that interleaves the topological repair and the geometric improvement of singularity graphs. In each iteration, the topological repair resolves locally non-meshable defects as depicted in Sec.7.3, and the subsequent geometric improvement smoothes the singularity graphs and removes redundant zipper nodes while aligning singularity graphs to frame fields. We begin with the topological correction of frame fields.

7.4.1 Frame Field Correction

With the toolbox of local repair operations described in Sec.7.3, the complete frame field correction follows the steps: (1) ensuring meshable edges; (2) repairing feature-feature polar sectors; (3) decomposing non-meshable singular vertices; (4) repairing constrained polar sectors; (5) fixing zipper nodes. The rationale behind this strategy is that all defects of the same type in the tet mesh are resolved in the current step. It may create new types of defects in the one-ring vicinity but can be fixed in the subsequent steps. We describe each step in detail below.

(1) *Ensuring meshable edges.* This step is to eliminate non-meshable edges due to compound monodromy and polar sector wedges in the tet mesh. For compound singular edges, two repair options are available: collapsing [LLX$^+$12] and splitting [JHW$^+$14]. Although the collapsing of an edge is not always possible due to edge collapsing conditions, the advantage is that the invalid vertex number will be decreased by one upon success. On the contrary, while the splitting approach ensures the elimination of a complex edge, the end vertices remain non-meshable. Therefore, in our practical algorithm, we encourage the complex

singular edges to be collapsed, except that edges touching the boundary are split instead of collapsing to the boundary. For polar sector wedges, we simply collect all involved edges and then apply the corresponding repair operation. After the step of ensuring meshable edges, adjacent singular vertices might become non-meshable and will be processed in step (3). Note that we also eliminate singular edges of high valence for better scaled Jacobian [LZC+18] unless they are unavoidable due to feature constraints.

(2) *Repair feature-feature polar sectors.* Feature-feature polar sectors are detected and repaired as in Sec.7.3.3. As a result, all sector angles are opened up by $\frac{\pi}{2}$, and additional zipper node (s) are created in the one-ring vicinity, which will be handled in step (5). Depending on the local feature configuration, it might also generate constrained polar sectors of type (2) or (3), as shown in Fig.7.11(b-c), which will be repaired in step (4).

(3) *Repair invalid singular vertices.* For invalid singular nodes incident to more than two singular edges which are not on the feature surface, we simply fix them in random order since the characterization of global meshability is not yet understood. It results in valid singular vertices or zipper nodes in the one-ring neighborhood. For invalid singular vertices on the feature surface, we maintain a processing queue to iteratively perform repair operations. Adjacent singular vertices might be invalidated afterward, as shown in Fig.7.15, and thus should be pushed into the queue for further processing. Since the solution to repairing singular vertices on arcs is unique but not for singular nodes, we first process singular vertices with fewer incident singular edges. Each repair reduces the number of the singular vertex on the feature surface by one, and the process terminates when the operation can fix no more. At the end of the step, the remaining non-meshable vertices are zipper nodes and the center vertices of constrained polar sectors.

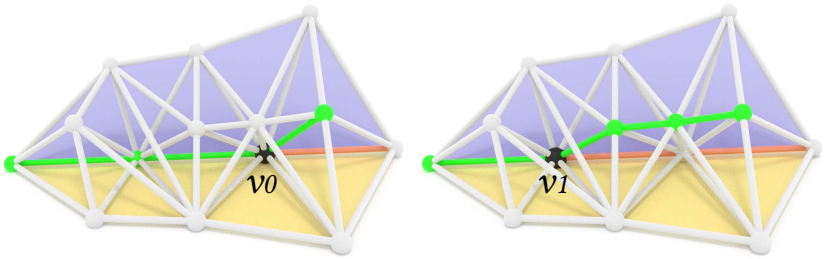


Figure 7.15: Repairing v_0 makes the adjacent singular vertex v_1 non-meshable.

(4) *Fix constrained polar sectors.* Constrained polar sectors only appear where feature edges are present in tet meshes. The detection of this class of defect only involves singular vertices on feature edges. As discussed in Sec.7.3.3, since the repair of constrained polar sectors of type (1) might introduce new ones of type (2), all ones of type (1) are processed first. We then check for constrained polar sectors of types (2) and (3) and repair them via the provided operations in Sec.7.3.3. The possibly created zipper node at the end vertex of the guiding arc will be fixed by the zipper node repair in step (5).

(5) *Fix zipper nodes.* Of all zipper nodes in frame fields, some are often unnecessary to fix due to geometric noise. Only inevitable zipper nodes are selected and fixed by unzipping. We iterate all singular arcs and take the furthest zipper node from the start vertex of the arc if the number of zipper nodes on the arc is odd. Besides, we fix zipper nodes with sufficiently long incident singular

segments. Unzipping results in pairs of singular arcs that end orthogonally on the boundary or connect to another inevitable zipper node that matches the direction. Although it likely brings in new zipper nodes because of zigzags, they can be resolved by singularity relocation and mesh improvement discussed in the next section.

7.4.2 Frame Field Aligned Singular Graph

One major issue in general frame field generators for hex meshing is that singular edges deviate from the frame field directions, corresponding to the defect (D1). Directly aligning frame fields to singular edges can induce high distortion due to noisy singularity graphs: (1) zigzag in input tet meshes, and (2) introduced by local repair operations. On the contrary, since the streamlines of frame fields exhibit a smoother structure, aligning singularity graphs to frame field directions is more natural. While improving the geometry of singularity graphs, it simultaneously reduces the singularity complexity by eliminating redundant zipper nodes.

In the generation of the new frame field $\mathcal{O}$ and its embedding tet mesh $\mathcal{T}$, several objectives are desired :

1. Meshable singular edge e aligns to the direction of the rotation axis u.
2. The tet mesh $\mathcal{T}$ contains no degenerate/flipped tetrahedra.
3. Features are preserved: vertex $v_i \in V_V$ is fixed, $v_i \in V_E$ moves along the feature arc and $v_i \in V_F$ stays on the feature surface. $V_F, V_F E, V_F F$ are sets of feature vertices, feature edge vertice and feature face vertices, respectively.
4. Shrink compound singular edges and singular edges at zipper nodes.

Optimizing positions of tet mesh vertices alone is insufficient to achieve all objectives. Keeping all tets valid is not guaranteed when the tet mesh lacks degrees of freedom in the vertex movement. Therefore, we alternate the tet mesh improvement and the singular vertex relocation in our algorithm.

Mesh Improving

The quality of tetrahedra incident to singularities can be rather poor after mesh modifications. A tet mesh improvement is required for two reasons: firstly, it creates necessary space for the singular vertex relocation in the next alternation; secondly, better tet mesh quality is beneficial for finding a valid seamless parameterization in the subsequent hex meshing pipeline. We adopt a similar strategy as in [HZG+18a], which in our case, consists of three passes: edge splitting, edge collapsing, and edge swapping. In our experiment, only the edges of the 2-ring tets of the singularity graph are involved in the remeshing.

Edge splitting. We assign the target edge length to each vertex and initialize it as the average length of its incident edges. In each splitting pass, edges of length $l_{e_{ij}} > \frac{5}{3}(tl_i + tl_j)$ are maintained in a priority queue and are split from the longest first.

Edge collapsing. Each half-edge $h_e = (v_f, v_t)$ with length longer than $\frac{3}{5}(tl_i + tl_j)$ is a candidate for collapsing. The order of edge collapsing pass is the

shortest the first. Aiming for mesh improvement, we check the quality of each tet t w.r.t. $tr(J_t^T J_t)/det(J_t)^{\frac{2}{3}}$, where J_t denotes the Jacobian which deforms t to a regular tet. We validate the collapsing if the worst element quality in the neighborhood will be improved. Besides, there are topological conditions when collapsing h_e is forbidden: (1) violating the link condition [LLX+12]; (2) h_e is not feature/singular but v_f is a feature/singular edge vertex; (3) it is not a feature face edge, but v_f is feature face vertex; (4) v_f is a feature vertex; (5) conflicting alignment constraints exist in the same chart after collapsing.

Edge swapping. In this pass, we perform 3-2, 4-4, and 5-6 edge swapping on regular edges in the 2-ring of singularities. Among all configurations of each edge swapping type, we find the best one w.r.t. the quality of the worst element. An edge swapping is accepted when the worst quality is improved. To maintain the *matchings* in the edge swapping, we need to locally transform the coordinate system in the charts incident to the edge e. This operation ensures that the matching on every incident face of e is $Identity$.

After every single mesh operation, all associated properties, including frame field properties, singularity properties, and feature properties, should be updated accordingly. In particular, we smooth frames without aligning to singularities in the involved chart after edge collapsing and swapping.

Singular Vertex Relocation

Let $x_i, p_i \in \mathbb{R}^3$ be the new/current position of the singular vertex v_i. The relocation of $v_i \in V_S$ can be posed as an optimization problem with linear constraints:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & E_a + E_s + E_c + E_d + E_r \\ \text{subject to} \quad & x_i - p_i = 0,\ \forall v_i \in V_V \\ & (x_i - p_i) \times t_i = 0,\ \forall v_i \in V_{FE} \\ & (x_i - p_i) \cdot n_i = 0,\ \forall v_i \in V_{FF}, \end{aligned}$$

where $t_i \in \mathbb{R}^3$ is the unit tangent vector of v_i on the feature arc and n_i is its unit normal on the feature surface. The details of each energy term are explained below.

Alignment energy E_a orients every meshable singular edge $e_{ij} \in E_h$ to the frame field direction d_{ij}. It is formulated as:

$$E_a = \frac{w_a}{s} \sum_{e_{ij} \in E_h} \frac{|(x_i - x_j) \times d_{ij}|^2}{l_{e_{ij}}}$$

with $d_{ij} \in \mathbb{R}^3$ being the average direction of rotation axes in the incident charts of e_{ij} and $l_{e_{ij}}$ being the edge length. It reaches the minimum when d_{ij} and $(x_i - x_j)$ are colinear.

Shrink energy serves for the objective (4) and is termed as:

$$E_s = \frac{w_s}{s} \sum_{e_{ij} \in E_C} \frac{|x_i - x_j|^2}{l_{e_{ij}}}$$

The contracted compound singular edges will be collapsed in the mesh improvement afterward. E_C is the set of all compound singular edges and singular edges incident to zipper nodes.

Curvature energy $E_c = \frac{w_c}{s} \sum_{e_{ij}, e_{jk} \in E_S} |x_i + x_k - 2x_j|^2 / (l_{e_{ij}} + l_{e_{jk}})$, acts as a regularizer of the alignment term, avoiding the undesired minimum where the singular edge aligns to the opposite direction of the desired. E_S is the set of meshable singular edges.

Deformation energy E_d maintains the validity of the tet mesh, preventing degenerate/flipped tets and guarding the surface normal against flipping. It is expressed as:

$$E_d = w_d \sum_{v_i \in V_S} \sum_{t \in T_{v_i}} \log \frac{tr(J_t^T J_t)}{det(J_t)^{\frac{2}{3}}} + w_s \sum_{v_i \in V_F} \sum_{f \in F_{v_i}} \log \frac{tr(J_f^T J_f)}{|det(J_f)|}$$

In the first term, V_S is the set of singular vertices, and T_{v_i} is the set of incident tets to the vertex v_i. When v_i is at the boundary, we consider a fan of virtual tets which takes the adjacent boundary faces as the base and $v_i' = v_i + tl_i * n_i$ as the tip vertex, tl_i being the target edge length at v_i and n_i being the vertex normal. The second term is the surface analog, where J_f denotes the Jacobian deforming f to a regular triangle. V_F contains singular vertices on the feature surface and F_{v_i} represents the set of feature triangles that are adjacent to v_i. The shape metrics are scale-invariant, and w_d, w_s are set to 0.001 and 0.05 in our experiment.

Repulsion energy $E_r = \frac{0.1 w_r}{s} \sum_{v_i \in V_r} E_{r_i}$ helps to prevent new singular arcs from getting too close after splitting zipper nodes. V_r contains vertices on the those singular arcs and E_{r_i} is expressed as:

$$E_{r_i} = \begin{cases} \frac{1}{2d_i} |x_i - \hat{p}_i|^2 & \text{if } d_i \leq tl_{min} \\ 0 & \text{otherwise} \end{cases}$$

Let p_i' be the closest point on other singular arcs to v_i, and d_i is the distance. $\hat{p}_i$ is the target point where v_i is pushed away to and is calculated as $\hat{p}_i = \frac{p_i + p_i'}{2} + tl_{min} * \frac{p_i - p_i'}{|p_i - p_i'|}$, where tl_{min} is the minimum target edge length of singular vertices on the same arc of v_i.

w_a, w_s, w_c, w_r are user-specified parameters set to 1 by default in our experiment. $s = 0.1 * \sqrt[3]{Vol_{\mathcal{C}}}$ is a constant for scale invariant, where $Vol_{\mathcal{C}}$ is the tet mesh volume. At the end of the singular vertex relocation, we smooth the frame field of 2-ring cells with alignment constraints of features except for singular edges. In the implementation, we use TinyAD [SBB+22] for automatic differentiation and CoMISo [BZK12] to solve the optimization problem.

7.4.3 Two-stage scheme

In this section, we describe the strategy that combines frame field correction, tet mesh improvement, and singularity relocation in our practical algorithm. Observing that all local repair operations in Sec.7.3 fix each type of defect might at the cost of bringing in zipper nodes, we advocate a two-stage scheme for the complete pipeline.

In the first stage, outlined in 6, we correct the frame field following the steps in Sec.7.4.1 except for zipper node repair, defined as *FrameFieldCorrectionExceptZipperNodes*, and then align the singularity graph to the frame field in each iteration with the method in Sec.7.4.2, called *AlignSingularGraphToFrameField*. We apply a local smoothing with feature alignment constraints of the frame field

in the 2-ring tets around singularities after the singularity relocation. Such an alternation is performed N_1 times in the first stage, where N_1 is a user-specified parameter (e.g., 8). Local defects other than zipper nodes can often be fixed in the first iteration. However, as our field is not aligned to the singularity at this point, the edge valence might flip the sign in a rather non-smooth field after the singular vertex relocation. Consequently, the signature of a singular node may become an invalid type due to the valence flipping of incident singular edges. It often takes a few iterations until the singularity graph reaches a relatively steady status. In addition, there might be compound singular edges that are not successfully collapsed in the first stage in rare cases. Splitting is employed to ensure the meshability of all edges.

Algorithm 6 FixNonMeshableFirstStage

Input: Tet mesh and frame field
Output: Tet mesh and frame field
1: **for** $i = 1$ to N_1 **do** ▷
2: FrameFieldCorrectionExceptZipperNodes() ▷
3: AlignSingularGraphToFrameField() ▷
4: **end for**

We follow a similar concept in the second stage but include the repair of zipper nodes in the frame field correction, named *FrameFieldCorrection*, summarized in Algo.7. As new singular arcs after unzipping are often zigzag, N_{inner} iterations of *AlignSingularGraphToFrameField* are applied afterward for singularity graph relocation, by default $N_{inner} = 6$. N_2 is the parameter of the iteration number of the second stage, which can optionally be stopped early once we detect that all feature and singular vertices are locally meshable. Fig.7.16 shows the status of the singularity graph of *n03c* in different stages. Note that smoothing out redundant zipper nodes might not always be possible in regions where several singular arcs intertwine after repairing zipper nodes. Alternatively, we can re-label the different valence of short segments and re-align relevant frames, called *UniformSingularArcValence*. Ultimately, we optimize the frame field with feature/singularity alignment constraints (*FrameFieldOptimization*).

Algorithm 7 FixNonMeshableSecondStage

Input: Tet mesh and frame field
Output: Tet mesh and frame field
1: **for** $i = 1$ to N_2 **do** ▷
2: FrameFieldCorrection() ▷
3: **for** $j = 1$ to N_{inner} **do**
4: AlignSingularGraphToFrameField() ▷
5: **if** isLocallyMeshable() **then** ▷
6: Stop ▷
7: **end if**
8: **end for**
9: **end for**
10: UniformSingularArcValence() ▷
11: FrameFieldOptimization() ▷

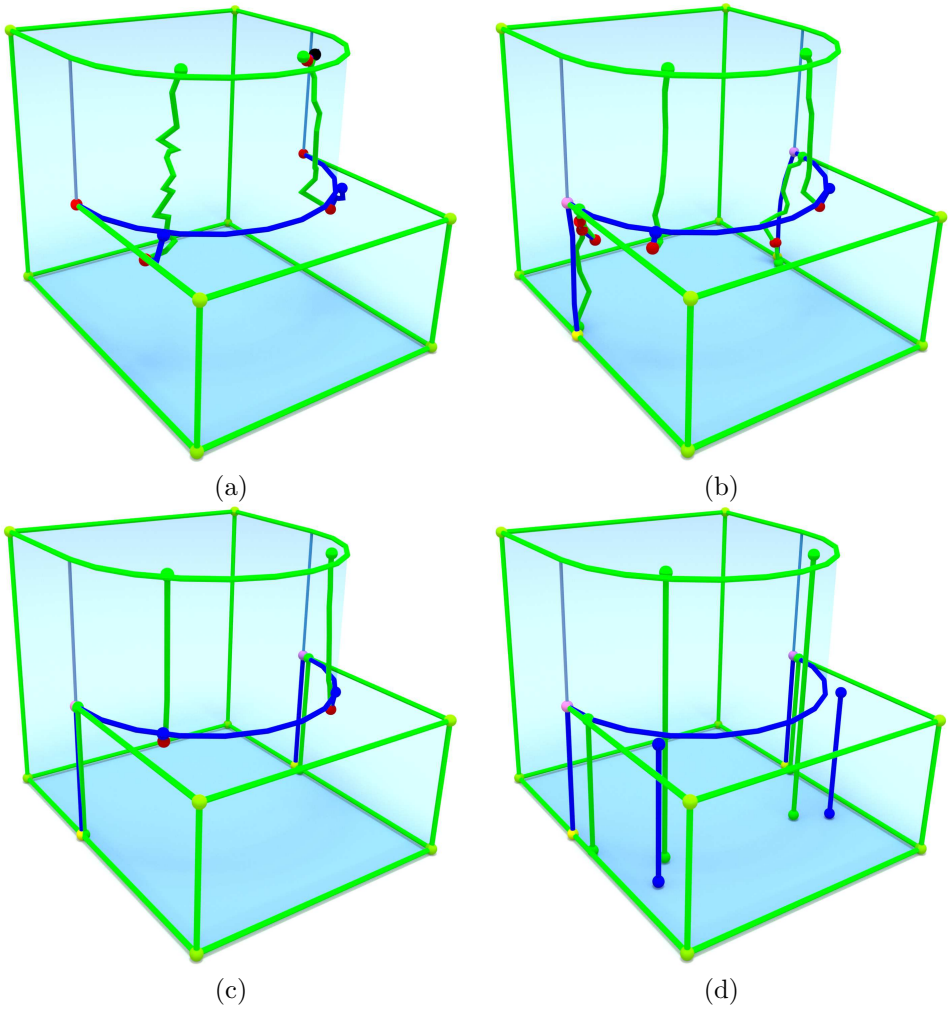


Figure 7.16: Algorithm Overview: (a) The initial frame field and its singularity graph. (b) Various local defects are repaired after the first iteration. (c) At the end of the first stage, only two zipper nodes are locally non-meshable. (d) The final singularity graph of the locally meshable frame field.

For the local meshability check, described in Sec. 7.6.1, we split the mesh if additional DOFs are required for compatible alignment constraints. As the input to parameterization, the final tet mesh needs further refinement to lower the frame field distortion since the field is piece-wise constant. We scale the target edge length according to the frame field distortion and then apply *mesh improving* to the whole tet mesh. Differently, we perform a compound *edge splitting* and *collapsing*, meaning the *middle vertex* in edge splitting and the *to vertex* in edge collapsing are allowed to move to the optimum position w.r.t. the tet quality.

7.5 Parameterization

7.5.1 Seamless Map

Given a locally meshable frame field $\mathcal{F}$ on tetrahedral cells and corresponding matchings M on half-faces, the next step in our hexahedral meshing framework consists in generating a locally injective seamless map $f : \mathbb{R}^3 \to \mathbb{R}^3$, which is needed for integer-quantization. A seamless map can be obtained by solving a Poisson problem in the spirit of [NRP11], including all feature alignment and cut constraints but ignoring integer constraints. Frequently, the result is a degenerate map with inverted elements, and improved robustness can be achieved by adding a post-process that targets local injectivity [GKK+21]. In our experiments, we often observed that [GKK+21] succeeds in finding a locally injective map but not being able to correctly reproduce the frame field topology. The issue is that the branched covering only dictates singular indices up to additional integers inside the domain and half-integers on the boundary, allowing e.g. a singular arc of index $-1\frac{1}{4}$ to change into one of index $\frac{3}{4}$, cf. Fig.13 of [GKK+22].

Integrable Frame Field Optimization To prevent such failure cases, we propose a novel seamless map optimization based on integrable frame fields and a smoothness regularizer. The main difference compared to the approach above is that we do not allow any inverted element throughout the optimization and, therefore, effectively prevent index changes that are typically caused by untangling inverted elements of the initial map. Hence, instead of starting the optimization from a conforming but potentially degenerate map, we do the opposite and start from a locally injective but non-conforming map. Dropping conformity is equivalent to giving up the integrability of the frame field. Hence we can simply initialize with any target frame field satisfying $\det F > 0$.

More precisely, for each tetrahedron t_i, we optimize a Jacobian matrix $J_i \in \mathbb{R}^{3\times3}$. Observing that we are targeting a map, which sends the tangent vectors of a frame onto coordinate axes, i.e., $J_i F_i = I$, the per-element initialization is simply $J_i = F_i^{-1}$. This initial map is locally injective since $\det F_i > 0$ and $\det J_i = 1/\det F_i$. Then we optimize a deformation objective with barrier behavior, tending to ∞ for degenerating elements. In all our experiment we use the symmetric Dirichlet energy $E_{SD} = \int_\Omega S_D(J)dV$ with

$$S_D(J) = \begin{cases} ||J||_2^2 + ||J^{-1}||_2^2 & \det J > 0 \\ \infty & \det J \leq 0 \end{cases} \tag{7.1}$$

Integrability in the sense of a piecewise linear map requires that the (matched) gradients of two neighboring tetrahedra t_i and t_j are identical when projected onto their common face f_{ij}, leading to linear constraints

$$P_{ij}(J_i^T - M_{ij}^{-T} J_j^T) = \bar{0} \tag{7.2}$$

with $P_{ij} \in \mathbb{R}^{2\times3}$ projecting gradients onto a basis of the plane of f_{ij}, $M_{ij} \in \mathcal{O}$ is a matching matrix from the octahedral group sending F_i to its representation $F_i M_{ij}$ in the chart of t_j, and $\bar{0} \in \mathbb{R}^{2\times3}$ a matrix of zeros.

Optimizing E_{SD} subject to linear integrability constraints and feature alignment constraints, often results in a locally injective seamless map. However, there is neither a guarantee on obtaining a locally injective map, nor on correctly re-producing the frame field topology. Due to the non-convex objective function, the optimization might converge to an infeasible point, never reaching an integrable frame field that would be equivalent to a conforming seamless map. Index changes, as discussed above, can also happen in this formulation. Therefore, to further improve correct re-production of the frame field topology, we add (1) a smoothness regularizer $E_S = \int_\Omega ||\nabla J||_2^2 dV$, and (2) ensure in the line search of the optimizer that we never pass through an inverted state – which is necessary to change the index assuming conformity. A candidate step $x + t\Delta x$ is truncated to a step length t such that $J + s\Delta J > 0$ for $s \in [0, t]$ is valid for all tetrahedra.

Implementation Details The integrable frame field is optimized with a Truncated Newton Method [WN+99] using a projected PCG linear solver to enable scalability to large tetrahedral meshes and accurate constraint satisfaction. All constraints feasible at the start are handled by the projected PCG, while integrability constraints, which are infeasible at the start, are included with quadratic penalty terms E_I, leading to the overall objective $E = E_{SD} + w_S E_S + w_I E_I$. As a final heuristic, which proved valuable in our experiments, we optimize a sequence of optimization problems, where we successively increase w_I and decrease w_S. All derivatives are computed algorithmically with [SBB+22], using Eigen [GJ+10].

7.5.2 Integer-Grid Map

Given a valid seamless map, we first obtain a valid quantization with the robust algorithm of [BBC22] using the motorcycle complex. The quantization constraints are then added to the optimization of the seamless map to compute a valid integer-grid map. Since the seamless map of the previous step serves as a locally injective initialization, we can immediately optimize a conforming piecewise linear map with the barrier energy E_{SD}. The linearly constrained non-convex problem is optimized with IPOPT [WB06]. In our experiments, the integer-grid map construction succeeded in all cases, where a valid seamless map was obtained. However, for extremely coarse quantizations, which require large geometric distortions, failures can be observed since the tetrahedral mesh might not even have sufficient degrees of freedom to admit a locally injective map subject to the quantization constraints.

7.6 Evaluation

We challenge our hex meshing framework with the HexMe dataset [BRK+22], which contains 189 tetrahedral meshes with feature tags carried over from CAD models and is specifically designed for consistent and practically meaningful evaluations of hex meshing algorithms. Our method generates valid Integer-Grid Maps for 109 models, demonstrating significant improvement in the robustness of the frame field-based hex meshing. We first compare with the state-of-the-art field-based hex meshing methods [LLX+12] and [JHW+14] in particular, on the

HexMe dataset. Then we analyze the results of our approach and make a more detailed discussion.

7.6.1 Comparisons

Since our method is the first that aims for locally meshable 3D frame fields, direct comparisons with other techniques are impossible. Two previous works on frame field singularity correction for hex meshing, [LLX+12] and [JHW+14], target for local meshability of edges via automatic edge collapsing and splitting, respectively. We compare our method with these two approaches regarding four aspects: local meshability of edges, local meshability of vertices, seamless map, and Integer-Grid map validity. Fig. 7.17 shows for all three approaches the percentage of models that are locally meshable at edges, locally meshable at vertices, reached a valid seamless map, or reached a valid integer-grid map.

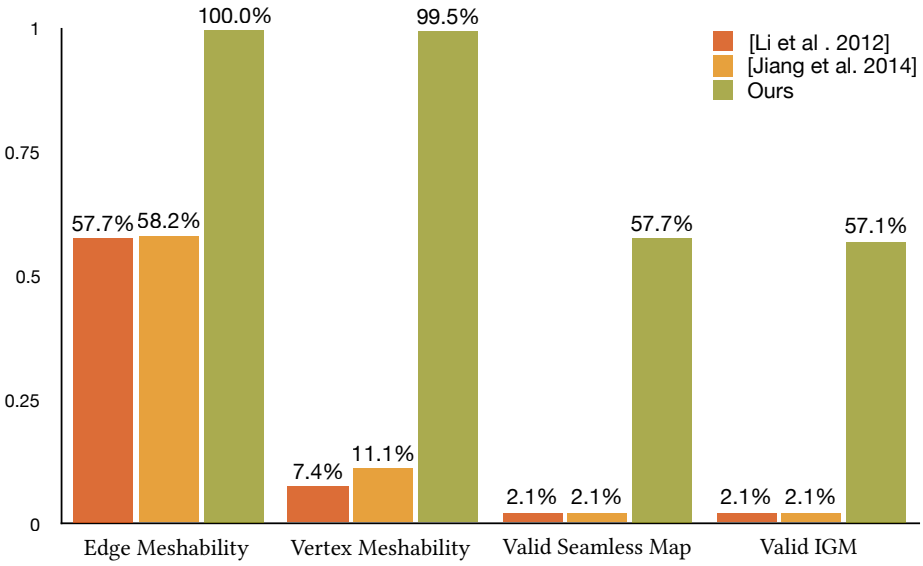


Figure 7.17: Comparison with [LLX+12] and [JHW+14] on HexMe dataset. From left to right, it shows the success rate of three methods w.r.t. edge meshability, vertex meshability, valid seamless map, and valid IGM.

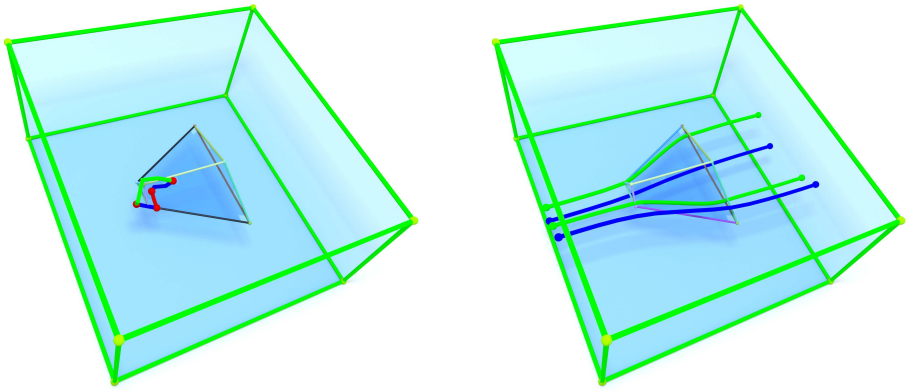
For fairness of the comparison, all three methods are run with identical octahedral fields, generated by [RSL16], with field alignment constraints to feature edges/faces. In the evaluation of local meshability, since all three methods involve mesh modification, counting the number of models that entirely pass each test is more meaningful than the number of non-meshable edges or vertices.

Edge Meshability Meshability of singular or feature edges is checked by measuring their index and requiring $index < 1$, i.e., at least one quad sector. For edges incident to feature surfaces, we also check that there is at least one quad sector in each feature sector.

The leftmost chart in Fig. 7.17 shows the numbers of models which pass the edge meshability test with three different methods. [LLX+12] and [JHW+14] succeed on 109 and 110 models respectively. Although all compound singular

arcs are repaired with these two methods, some feature edges with polar sectors remain non-meshable. Another typical defect is that singular arcs touch the boundary tangentially, inevitably leading to non-meshable polar sectors. Our method achieves local edge meshability for all 189 models. Fig.7.18(a)/(b) shows the singularity graphs of *n04b_transition_prism* after the correction with other/our methods.

Vertex Meshability The local meshability of frame fields is examined by testing whether the one-ring charts incident to each vertex of the tet mesh contains no invalid sectors. We employ the seamless map construction of Sec. 7.5.1 restricted to the one-ring neighborhood of a vertex in order to explicitly verify meshability. While [LLX+12] and [JHW+14] obtain 14 and 21 successful results, respectively, 188 out of 189 models pass the vertex meshability test with our approach. The behavior of the other two methods is expected since they lack the correction of invalid singular and polar sector wedges. In our test, the sole missing case is the tire model *i28b_gc_tire_1218*, which exceeded our time limit of 48h due to a huge number of zipper points in the frame field.



(a) (b)

Figure 7.18: Local meshability. (a) Singularity graph after correction with [LLX+12] and [JHW+14]. (b) Our singularity graph. Red edges on the interior feature surface, and black edges are not locally meshable. Green and blue edges represent singular edges of index $-\frac{1}{4}$ and $\frac{1}{4}$, and zipper nodes are colored in red. Other colored edges are features.

Seamless Map Besides only ensuring local meshability, our novel pipeline significantly improves hex meshing results. We first measure the number of valid locally injective seamless maps with correctly re-produced frame field sectors. In the experiment, we parameterize the output frame fields of [LLX+12] and [JHW+14] with the standard method described in [NRP11]. In comparison, ours uses the advanced parameterization technique, which effectively avoids degenerated and flipped tetrahedra. As shown in Fig. 7.17, our method successfully generates 109 valid seamless maps, while the other two approaches only succeed in four cases. The construction of valid seamless maps verifies that many of our locally meshable fields are also globally meshable.

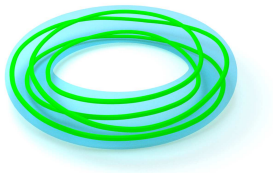
Integer-Grid Map We then compare the number of valid integer-grid maps generated by the three methods. The three bars on the right side in Fig. 7.17 demonstrate the number of valid IGMs that each method achieves. The two competing techniques achieve four valid IGMs, respectively, while ours generate 109 valid IGMs. The final hex meshes are extracted from IGMs with HexEx [LBK16]. Note that since HexEx can fix some local defects in the parameterization, the other methods may produce a few more valid hex meshes despite invalid IGMs.

7.6.2 Results and Discussion

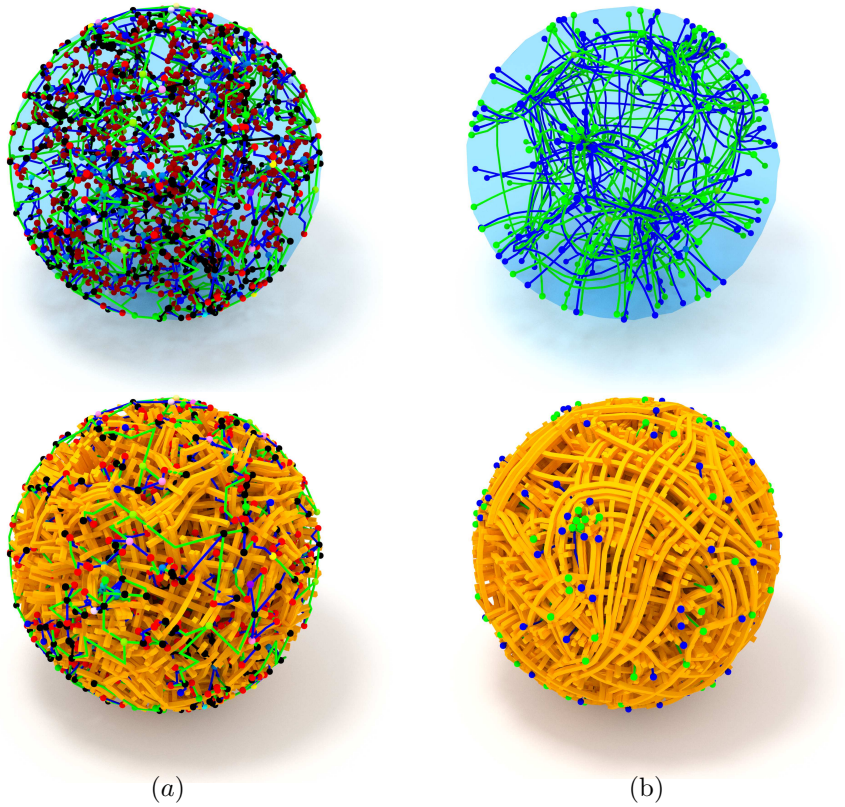
Local Meshability We refer to local meshability as vertex meshability since a locally meshable vertex implies that all incident edges are meshable. As stated in the comparisons on the HexMe dataset, nearly all output frame fields of the 189 meshes are locally meshable except for *i28b_gc_tire_1218*.

As a stress test, we run our pipeline on the tet mesh of a sphere with a random initial field instead of a smooth frame field [RSL16], and it successfully outputs a locally meshable frame field. The initial field is created in the experiment by sampling randomized unit quaternions to tet mesh vertices. Fig. 7.19 depicts the singularity graphs and streamlines of the random field and the locally meshable octahedral field. Note that for singular nodes colored other than red and black in Fig. 7.19(a), although hex mesh nodes with the same signatures exist [LZC+18], they are not necessarily locally meshable. Our method can correct all local defects in the random field and deliver a locally meshable field, see Fig. 7.19(b).

Global Meshability The global meshability is measured in terms of valid seamless parameterization. Our method successfully achieved 109 valid seamless maps out of 189 meshes. Fig. 7.20 reveals the performance on each category of the HexMe dataset. The top three bars of Fig. 7.20 show the number of valid (olive) vs.invalid (orange) seamless maps of the curvature-adapted/uniform/box-embedded meshes. As our local repair operations involve splitting, the mesh resolution does not influence the hex meshing results much. It delivers 68.3%/66.7% success rate for curvature-adapted/uniform meshes, respectively. However, in a few cases, different discretization leads to different singularity graphs of the input frame fields, which can be globally non-meshable. For example, the initial singularity graph of *s16u_torus* has twists that are not repairable with local operations while *s16c_torus* is meshable. The box-embedded category, which has interior features, is the most challenging for the pipeline since the global structure of singularity graphs is topologically more complex. The number drops to 38.1% in this class.



The three bars at the bottom indicate the performance of the hex meshing pipeline on the industrial/nasty/simple models. It obtains 36.6% valid seamless maps for industrial meshes, and the number increases to 60.0% for the meshes in the nasty category. As expected, the pipeline performs the best on the simple models, achieving 94.1% success rate. Note that excluding the box-embedded



(a) (b)

Figure 7.19: Stress test. (a) A random field and its singularity graph. (b) The output field and its singularity graph. Black edges are complex singular edges, and black vertices are invalid singular nodes.

meshes, the success rate of the three categories goes up to 48.4% / 73.3% / 97.1%, and the numbers are shown in the middle three bars in Fig. 7.20.

Fig. 7.21 exhibits a subset of the output hex meshes, and the complete hex meshes are attached in the supplemental material. We parameterize the tet meshes for seamless maps and then re-parameterize it with integer constraints via a robust quantization [BBC22] for IGMs. We finally extract hex meshes with HexEx [LBK16]. The input feature entities in the tet meshes are preserved in the output hex meshes shown in different colors. Fig. 7.21(g-k) demonstrates the capability of our method in hex meshing models with interior features. In the post-processing, we improve the geometric quality of hex meshes using standard wrappers provided in [BDK$^+$03]. Statistics on minimum/average scaled Jacobian of the shown hex meshes can be found in table 7.1.

Timing In our practical algorithm, the generation of locally meshable frame fields (frame field repair) is time-efficient. The procedures of repairing local defects of frame fields involve only local operations. The singularity relocation step optimizes only the position of singular vertices and improves the $2-$ring tet meshes of singularity graphs. However, the downstream steps in the hex meshing pipeline, the integrability optimization of frame fields, and the param-

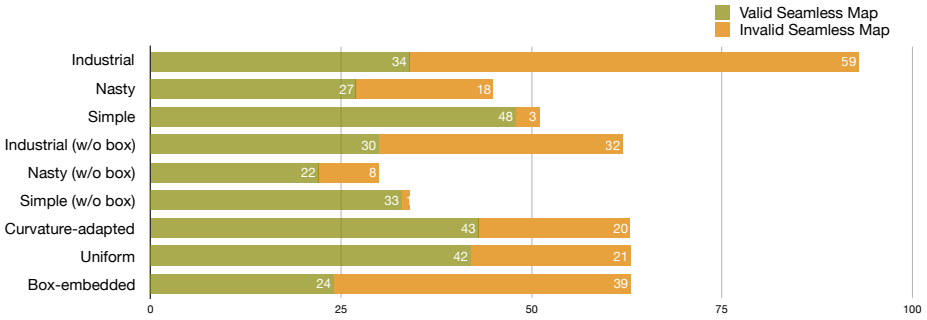
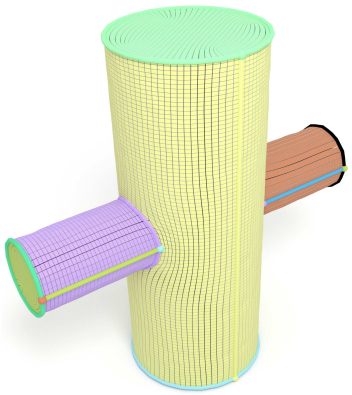


Figure 7.20: Number of valid vs.invalid seamless maps obtained in each category.

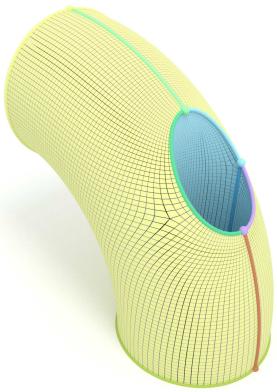
Mesh	#Tet	Scaled Jacobian	Frame Field Repair	Frame Field Opt.	Param. (s)
i06u_m6	189547	0.218/0.977	73	1992	5716
i12u_s5	107154	0.038/0.963	52	974	1085
i14b_s7	70968	0.133/0.954	327	1440	4097
i15b_s8	132749	0.268/0.973	278	779	5785
i18b_s22	295584	0.132/0.969	451	3373	41475
i25c_s40	24305	0.126/0.958	16	195	434
n08c_pentapyr	6144	0.269/0.926	31	166	101
n09c_pyramid	7015	0.264/0.948	10	39	30
n10u_qtorus_cyl	67154	0.307/0.974	279	829	536
s04b_tetrahedron	36329	0.123/0.973	23	93	1059
s08c_cross_cyls_dr	19747	0.231/0.982	36	91	84

Table 7.1: Statistics and Timings.

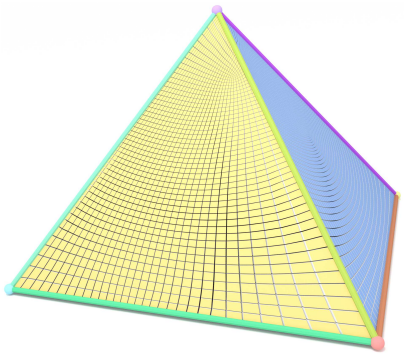
eterization are much more expensive. Table 7.1 summarizes the time cost on example meshes with complexity from 6k to 300k tetrahedra. The experiment is done with a single thread of a cluster with CPU AMD EPYC 7742. The time costs of the frame field repair range from 10 to 451 seconds, while it takes up to 3373 seconds in the frame field optimization. The parameterization, in most cases, dominates the time cost. Since we do not aim for singularity graphs with optimum geometry regarding the integrability energy, they may induce frame fields with large distortion which are challenging for optimization. Similarly, the final tet mesh may contain tets of poor quality due to constraints imposed by sharp features and singularity graphs, and is thus challenging for the parameterization.



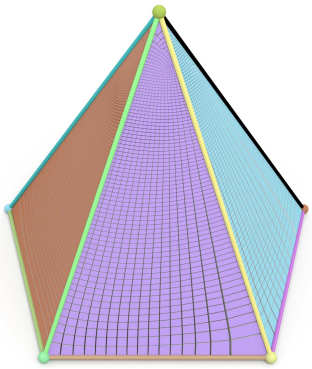
(a) `s08c_cross_cyls_dr`



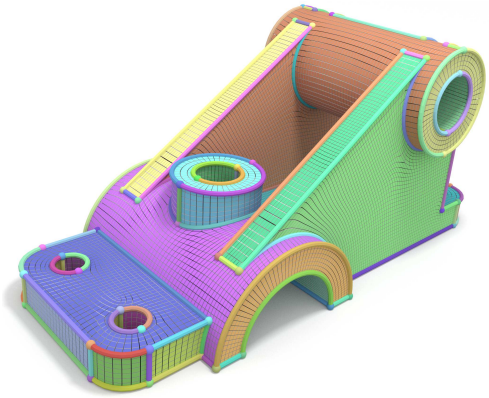
(b) `n10u_qtorus_cyl`



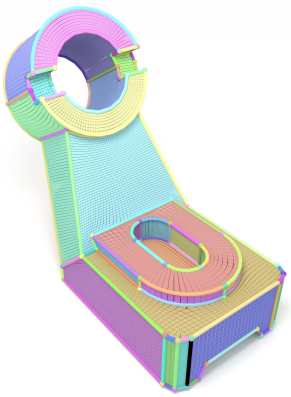
(c) `n09c_pyramid`



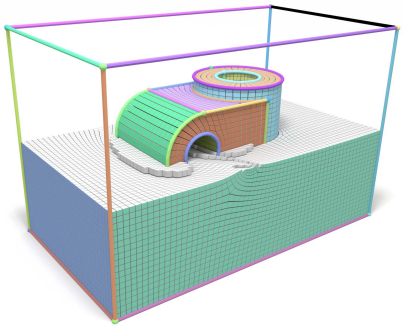
(d) `n08c_pentapyr`



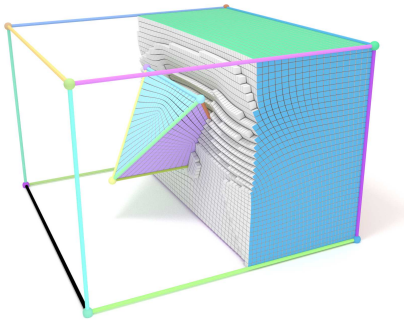
(e) `i12u_s5`



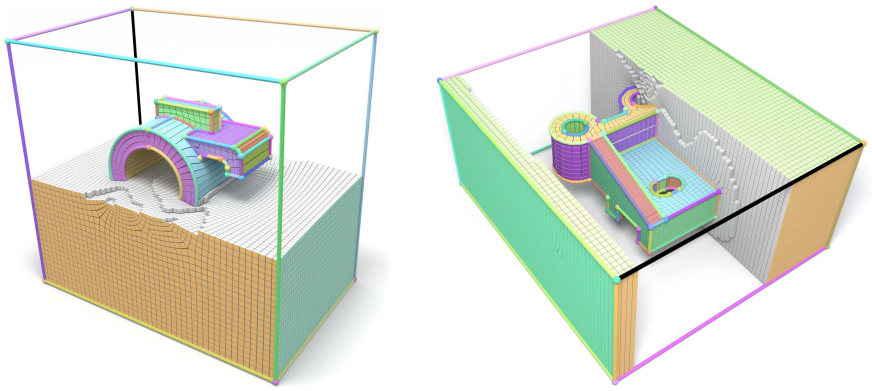
(f) `i25c_s40`



(h) `i14b_s7`



(g) `s04b_tetrahedron`



(i) `i15b_s8` (k) `i18b_s22`

Figure 7.21: Several hex meshes automatically generated by our method. Feature nodes/arcs/surfaces are colored and preserved as in the input tet meshes.

Discussion For the local meshability test, we use default parameters for all meshes. For hex meshing tasks, the pipeline outputs globally meshable frame fields for most tested meshes with the same parameters, while it needs parameter tuning for a few meshes. The main issue comes from the zipper node repair. The dual path might end at an undesirable boundary or zipper node, leading to non-meshable singularity graph. Additionally, the singular arc pair might be severely twisted, which causes failures in finding a feasible solution in the downstream parameterization. We can adjust the zipper nodes' positions with the field alignment weight and the number of iterations of the first stage. For a few models, we allow merging zipper nodes to orthogonal singular arcs instead of *unzipping*.

Although our primary focus is on the local meshability of frame fields, the hex meshing pipeline performs well on global meshability, outputting 57.7% valid IGMs. Since our algorithm only involves local repair operations, global defects remain invalid, e.g., the twist of singular arcs in the initial frame fields. Starting from better frame fields such as [PBS20] can help to generate more valid hex meshes. Another aspect is that we might end up in globally non-meshable configurations in repairing invalid singular nodes. Multiple choices for determining parallel singular arcs could be locally feasible but not globally. Exploring sufficient conditions for the global meshability of frame fields will be left for future work.

Chapter 8

Conclusion

Of all major parts of the frame field base hex meshing: (i) generation of a feature-aligned frame field, (ii) generation of a seamless map, (iii) integer quantization, (iv) generation of the integer-grid map, and (v) extraction of (potentially higher-order) hexahedral mesh, provably robust algorithms exist only for the integer quantization (iii) [BBC22], and the (linear) mesh extraction (v)[LBK16], while steps (i),(ii), and (iv) remain fragile. This thesis focuses on understanding fundamental problems in frame fields regarding meshability and development of practical algorithms to enhance the robustness of the field based hex mesh generation pipeline. We will first summarize the main contributions of this thesis and then discuss the limitations and potential directions for future work.

8.1 Summary

The most fundamental question is why a feature-aligned smooth frame field often does not induce a valid hex mesh. From the singularity point of view, we show that only 11 configurations exist at hex mesh vertices if the edge valence of hex meshes is bounded, while many more invalid configurations can happen in smooth frame fields. This leads to the local and global necessary conditions of meshability w.r.t. singularity graphs of frame fields. With a locally meshable singularity graph as input, we present an algorithm that reconstructs the corresponding smooth frame field. It allows users to provide as input a self-designed singularity graph and (assuming global meshability) obtain a hex mesh with the same topology as output.

Towards practical applications, feature preservation must be considered in the automatic field based hex meshing pipeline. Together with colleagues, we generated the HexMe dataset consisting of representative meshes with feature tags which enable a comprehensive evaluation of field based hex meshing algorithms. Evaluating the standard field based hex meshing pipeline reveals that the current automatic approach is too fragile for practical applications.

Most importantly, we extend local meshability conditions to frame fields with feature constraints. In this part, we first analyze the topological structures that exist in general 2D frame fields but not in quad mesh-induced frame fields and identify the necessary and sufficient conditions for meshability of frame fields in 2D. We then study the meshability of singularity arcs and singular nodes in 3D

frame fields. In particular, singular nodes in 3D frame fields are decomposable into fundamental pieces. A series of repair operations of local defects are proposed with the help of *arc zipping*. Finally, we present a novel algorithm that converts a given frame field into a locally meshable one, significantly improving the robustness of field based hex mesh generation algorithms. Besides, a more robust algorithm than available state-of-the-art techniques is designed to optimize seamless maps and integer-grid maps for a given locally meshable frame field.

8.2 Limitations and Future Work

Automation For the singularity-constrained octahedral frame fields, a globally meshable singularity graph embedded in a tet mesh is required to get a valid hex mesh. However, there is no complete characterization of the global meshability requirements of singularity graphs. It relies on users' expertise to design a meshable singularity graph which is sometimes challenging on complex models. Future work in this direction is to automatize the generation of a globally meshable singularity graph suitable to the model. One way might be to automatically decompose models into more straightforward templates whose singularity structures are known and then glue individual pieces together. Another potential direction might be to *learn* the maps from singularity graphs to shapes.

Robustness Although our locally meshable frame fields significantly improve the robustness of the field based hex meshing pipeline, a valid output hex mesh is still far from being guaranteed due to the missing conditions of global meshability. A local repair operation, e.g., decomposing singular nodes, might end up with a non-meshable frame field if a wrong choice is made despite being locally correct, as shown in Fig. 8.1. In repairing zipper nodes, the two edge paths might be twisted as in in Fig. 8.1(c), which can be too difficult to find a locally injective map. Incorporating the *tangent point energy* as in [YSC21] could be effective in untangling the twists. Another intriguing direction is to relocate singularities by energy which reflects the distortion of the integer grid maps. This will not only untangle the twists but also move the singularities to better location.

Since local meshability is necessary but not sufficient for global meshability, the obvious next step will be to target full meshability. A direct generalization of [MPZ14] is impossible due to the non-integrability of 3D frame fields' surfaces. However, developing a method to generate the motorcycle complex [BGMC22] directly from a frame field is nevertheless a promising research direction. Such a method would inevitably require additional repair mechanisms since, in contrast to the 2D setting, singularity constraints and feature constraints can prevent the existence of a valid motorcycle complex, even for a locally meshable frame field.

Future work is also required to obtain stronger guarantees in the construction of locally injective maps. Even if a meshable frame field is available, none of the existing techniques can guarantee finding a seamless map of identical topology. Simply relocating the vertex position is not sufficient in cases where it requires

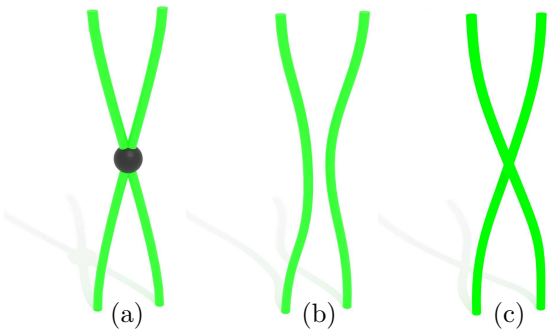


Figure 8.1: When decomposing an invalid singular node (a), detaching the correct singular arc results in a globally meshable frame field (b). At the same time, the other choice might have an impact on global meshability (c).

more degrees of freedom to avoid degeneracies/foldovers. Mesh splitting might be necessary when such scenarios are detected.

Quality Our locally meshable frame fields are not necessarily optimum in terms of the quality of output hex meshes since the distortion is not considered in the formulation. For example, the locations of the interior singular arc of the prism are equally good as long as it is aligned to the field direction in our locally meshable frame field, while the optimum location is expected to be the center of the volume for the best geometric quality of the hex mesh. In future work, we can devise such a distortion-driven energy to relocate singularity graphs of locally meshable frame fields. Besides geometric relocation, topological optimization might be another direction worth exploring. With the fixed singularity graph, simply adjusting its position may not be sufficient to achieve the best quality. We can further reduce the distortion by applying singular arc *zipping* and *unzipping* at areas of high distortion. This approach is more flexible and effective than [CAS+19, GLYL20] since it is not restricted to polycube maps.

Control A key requirement of practical hex meshing algorithms is to offer control of hex mesh properties, such as size, orientation, and anisotropy. An early approach [XGDC17] achieves adaptivity by magnifying small-scale areas with a scale factor, hex meshing the deformed region, and finally mapping it back to the original domain. The promising approach [FHTB21] provides more flexible controls over sizing, shear, and orientation with metric-driven frame fields. However, it suffers from the issues of non-meshable frame fields as most field based hex meshing approaches do. Incorporating our local repair operations might significantly improve the success rate.

Towards practical applications, frame field based hex meshing algorithms should also support hard constraints, e.g., pre-meshed sub-regions of the volumetric domain. Future work needs to consider these constraints in the frame field generation and quantization of parametrization such that individual pieces conformally stitch together.

Scalability The ultimate goal of the field based hex meshing pipeline is to mesh general shapes of varying complexity in a reasonable amount of time. Developing parallelization strategies for our pipeline is crucial to achieving scalability. This requires designing a challenging synchronization scheme among individual components, which depends on previous steps in the entire hex meshing pipeline. Parallelizing each step, however, is more reachable in the near future. For example, the tet mesh improvement which dominates the time cost of our locally meshable frame field generation can be parallelized straightforwardly [HSW+20].

Bibliography

[AFTR15] Armstrong, Cecil G. ; Fogg, Harold J. ; Tierney, Christopher M. ; Robinson, Trevor T.: Common themes in multi-block structured quad/hex mesh generation. In: *Procedia Engineering* 124 (2015), S. 70–82

[Alt22] *HyperMesh*. `https://www.altair.com/hypermesh/`. Version: 2022

[ANS22] *ANSYS*. `https://www.ansys.com/`. Version: 2022

[Asi93] Asimov, Daniel: Notes on the topology of vector fields and flows / Technical report, NASA Ames Research Center, 1993. RNR-93-003. 1993. – Forschungsbericht

[BBC22] Brückler, Hendrik ; Bommes, David ; Campen, Marcel: Volume parametrization quantization for hexahedral meshing. In: *ACM Transactions on Graphics (TOG)* 41 (2022), Nr. 4, S. 1–19

[BCE+13] Bommes, David ; Campen, Marcel ; Ebke, Hans-Christian ; Alliez, Pierre ; Kobbelt, Leif: Integer-grid maps for reliable quad meshing. In: *ACM Transactions on Graphics (TOG)* 32 (2013), Nr. 4, S. 1–12

[BDK+03] Brewer, Michael L. ; Diachin, Lori F. ; Knupp, Patrick M. ; Leurent, Thomas ; Melander, Darryl J.: The Mesquite Mesh Quality Improvement Toolkit. In: *IMR*, 2003

[BERF08] Brodersen, Olaf ; Eisfeld, Bernhard ; Raddatz, Jochen ; Frohnapfel, Petra: DLR results from the third AIAA computational fluid dynamics drag prediction workshop. In: *Journal of Aircraft* 45 (2008), Nr. 3, S. 823–836

[BGMC22] Brückler, Hendrik ; Gupta, Ojaswi ; Mandad, Manish ; Campen, Marcel: The 3D Motorcycle Complex for Structured Volume Decomposition. In: *Computer Graphics Forum* Bd. 41 Wiley Online Library, 2022, S. 221–235

[BLP+13] Bommes, David ; Lévy, Bruno ; Pietroni, Nico ; Puppo, Enrico ; Silva, Claudio ; Tarini, Marco ; Zorin, Denis: Quad-mesh generation and processing: A survey. In: *Computer Graphics Forum* Bd. 32 Wiley Online Library, 2013, S. 51–76

[BRK+22] Beaufort, P.-A. ; Reberol, M. ; Kalmykov, D. ; Liu, H. ; Ledoux, F. ; Bommes, D.: Hex Me If You Can. In: *Computer Graphics Forum* 41 (2022), Nr. 5, 125-134. http://dx.doi.org/https://doi.org/10.1111/cgf.14608. – DOI https://doi.org/10.1111/cgf.14608

[BRM+14] Baudouin, Tristan C. ; Remacle, Jean-François ; Marchandise, Emilie ; Henrotte, François ; Geuzaine, Christophe: A frontal approach to hex-dominant mesh generation. In: *Advanced Modeling and Simulation in Engineering Sciences* 1 (2014), Nr. 1, S. 1–30

[BTP+19] Bracci, Matteo ; Tarini, Marco ; Pietroni, Nico ; Livesu, Marco ; Cignoni, Paolo: HexaLab.net: An online viewer for hexahedral meshes. In: *Computer-Aided Design* 110 (2019), S. 24–36. – ISSN 0010–4485

[BZK09] Bommes, David ; Zimmer, Henrik ; Kobbelt, Leif: Mixed-integer quadrangulation. In: *ACM Transactions On Graphics (TOG)* 28 (2009), Nr. 3, S. 1–10

[BZK12] Bommes, David ; Zimmer, Henrik ; Kobbelt, Leif: Practical mixed-integer optimization for geometry processing. In: *Curves and Surfaces: 7th International Conference, Avignon, France, June 24-30, 2010, Revised Selected Papers 7* Springer, 2012, S. 193–206

[Car02] Carey, Graham F.: Hexing the tet. In: *Communications in Numerical Methods in Engineering* 18 (2002), Nr. 3, S. 223–227

[CAS+19] Cherchi, Gianmarco ; Alliez, Pierre ; Scateni, Riccardo ; Lyon, Max ; Bommes, David: Selective padding for polycube-based hexahedral meshing. In: *Computer graphics forum* Bd. 38 Wiley Online Library, 2019, S. 580–591

[CBK15] Campen, Marcel ; Bommes, David ; Kobbelt, Leif: Quantized global parametrization. In: *Acm Transactions On Graphics (TOG)* 34 (2015), Nr. 6, S. 1–12

[CC19] Corman, Etienne ; Crane, Keenan: Symmetric moving frames. In: *ACM Transactions on Graphics (TOG)* 38 (2019), Nr. 4, S. 1–16

[CDS10] Crane, Keenan ; Desbrun, Mathieu ; Schröder, Peter: Trivial connections on discrete surfaces. In: *Computer Graphics Forum* Bd. 29 Wiley Online Library, 2010, S. 1525–1533

[CFG+15] Chang, Angel X. ; Funkhouser, Thomas ; Guibas, Leonidas ; Hanrahan, Pat ; Huang, Qixing ; Li, Zimo ; Savarese, Silvio ; Savva, Manolis ; Song, Shuran ; Su, Hao u. a.: Shapenet: An information-rich 3d model repository. In: *arXiv preprint arXiv:1512.03012* (2015)

[CHRS18] Chemin, Alexandre ; Henrotte, François ; Remacle, Jean-François ; Schaftingen, Jean V.: Representing three-dimensional cross fields using fourth order tensors. In: *International Meshing Roundtable* Springer, 2018, S. 89–108

[CLS16] Cherchi, Gianmarco ; Livesu, Marco ; Scateni, Riccardo: Polycube simplification for coarse layouts of surfaces and volumes. In: *Computer Graphics Forum* Bd. 35 Wiley Online Library, 2016, S. 11–20

[Cor22] *CoreForm*. `https://coreform.com/`. Version: 2022

[CRBH06] Cottrell, J A. ; Reali, Alessandro ; Bazilevs, Yuri ; Hughes, Thomas J.: Isogeometric analysis of structural vibrations. In: *Computer methods in applied mechanics and engineering* 195 (2006), Nr. 41-43, S. 5257–5296

[CUB22] *CUBIT*. `https://cubit.sandia.gov/`. Version: 2022

[CZ17] Campen, Marcel ; Zorin, Denis: Similarity maps and field-guided T-splines: a perfect couple. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–16

[DAZ+20] Du, Xingyi ; Aigerman, Noam ; Zhou, Qingnan ; Kovalsky, Shahar Z. ; Yan, Yajie ; Kaufman, Danny M. ; Ju, Tao: Lifting simplices to find injectivity. In: *ACM Trans. Graph.* 39 (2020), Nr. 4, S. 120

[EBCK13] Ebke, Hans-Christian ; Bommes, David ; Campen, Marcel ; Kobbelt, Leif: QEx: Robust quad mesh extraction. In: *ACM Transactions on Graphics (TOG)* 32 (2013), Nr. 6, S. 1–10

[EE14] Elsheikh, Ahmed H. ; Elsheikh, Mustafa: A consistent octree hanging node elimination algorithm for hexahedral mesh generation. In: *Advances in Engineering Software* 75 (2014), S. 86–100

[EPOM11] Ebeida, Mohamed S. ; Patney, Anjul ; Owens, John D. ; Mestreau, Eric: Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. In: *International Journal for Numerical Methods in Engineering* 88 (2011), Nr. 10, S. 974–985

[Eri13] Erickson, Jeff: Efficiently hex-meshing things with topology. In: *Proceedings of the twenty-ninth annual symposium on Computational geometry*, 2013, S. 37–46

[FBL16] Fu, Xiao-Ming ; Bai, Chong-Yang ; Liu, Yang: Efficient volumetric polycube-map construction. In: *Computer Graphics Forum* Bd. 35 Wiley Online Library, 2016, S. 97–106

[FHTB21] Fang, Xianzhong ; Huang, Jin ; Tong, Yiying ; Bao, Hujun: Metric-driven 3D frame field generation. In: *IEEE Transactions on Visualization and Computer Graphics* (2021)

[FP09] Fabri, Andreas ; Pion, Sylvain: CGAL: The computational geometry algorithms library. In: *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2009, S. 538–539

[FSZ+21] Fu, Xiao-Ming ; Su, Jian-Ping ; Zhao, Zheng-Yu ; Fang, Qing ; Ye, Chunyang ; Liu, Ligang: Inversion-free geometric mapping construction: A survey. In: *Computational Visual Media* 7 (2021), Nr. 3, S. 289–318

[FXBH16] Fang, Xianzhong ; Xu, Weiwei ; Bao, Hujun ; Huang, Jin: All-hex meshing using closed-form induced polycube. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 4, S. 1–9

[GBR21] Günther, Tobias ; Baeza Rojo, Irene: Introduction to vector field topology. In: *Topological methods in data analysis and visualization vi*. Springer, 2021, S. 289–326

[GDC15] Gao, Xifeng ; Deng, Zhigang ; Chen, Guoning: Hexahedral mesh re-parameterization from aligned base-complex. In: *ACM Transactions on Graphics (TOG)* 34 (2015), Nr. 4, S. 1–10

[GHX+17] Gao, Xifeng ; Huang, Jin ; Xu, Kaoji ; Pan, Zherong ; Deng, Zhigang ; Chen, Guoning: Evaluating Hex-mesh Quality Metrics via Correlation Analysis. In: *Computer Graphics Forum* Bd. 36 Wiley Online Library, 2017, S. 105–116

[GJ+10] Guennebaud, Gaël ; Jacob, Benoît u. a.: *Eigen v3*. http://eigen.tuxfamily.org, 2010

[GJTP17] Gao, Xifeng ; Jakob, Wenzel ; Tarini, Marco ; Panozzo, Daniele: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1–13

[GKK+21] Garanzha, Vladimir ; Kaporin, Igor ; Kudryavtseva, Liudmila ; Protais, François ; Ray, Nicolas ; Sokolov, Dmitry: Foldover-free maps in 50 lines of code. In: *ACM Transactions on Graphics (TOG)* 40 (2021), Nr. 4, S. 1–16

[GKK+22] Garanzha, Vladimir ; Kaporin, Igor ; Kudryavtseva, Liudmila ; Protais, François ; Desobry, David ; Sokolov, Dmitry: Practical lowest distortion mapping. In: *arXiv preprint arXiv:2201.12112* (2022)

[GLYL20] Guo, Hao-Xiang ; Liu, Xiaohan ; Yan, Dong-Ming ; Liu, Yang: Cut-enhanced PolyCube-maps for feature-aware all-hex meshing. In: *ACM Transactions on Graphics (TOG)* 39 (2020), Nr. 4, S. 106–1

[GMS21] Golovaty, Dmitry ; Montero, Jose A. ; Spirn, Daniel: A variational method for generating n-cross fields using higher-order Q-tensors. In: *SIAM Journal on Scientific Computing* 43 (2021), Nr. 5, S. A3269–A3304

[GP21] Gao, Ruiliang ; Peters, Jörg: Improving Hexahedral-FEM-Based Plasticity in Surgery Simulation. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention* Springer, 2021, S. 571–580

[GPW+17] Gao, Xifeng ; Panozzo, Daniele ; Wang, Wenping ; Deng, Zhigang ; Chen, Guoning: Robust structure simplification for hex re-meshing. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 6, S. 1–13

[GR09] Geuzaine, Christophe ; Remacle, Jean-François: Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. In: *International journal for numerical methods in engineering* 79 (2009), Nr. 11, S. 1309–1331

[GSP19] Gao, Xifeng ; Shen, Hanxiao ; Panozzo, Daniele: Feature Preserving Octree-Based Hexahedral Meshing. In: *Computer graphics forum* Bd. 38 Wiley Online Library, 2019, S. 135–149

[GSZ11] Gregson, James ; Sheffer, Alla ; Zhang, Eugene: All-hex mesh generation via volumetric polycube deformation. In: *Computer graphics forum* Bd. 30 Wiley Online Library, 2011, S. 1407–1416

[HJS+14] Huang, Jin ; Jiang, Tengfei ; Shi, Zeyun ; Tong, Yiying ; Bao, Hujun ; Desbrun, Mathieu: l1-based construction of polycube maps from complex shapes. In: *ACM Transactions on Graphics (TOG)* 33 (2014), Nr. 3, S. 1–11

[HL88] Ho-Le, K.: Finite Element Mesh Generation Methods: A Review and Classification. In: *Computer Aided Design* 1 (20) (1988), S. 27–38

[HPM02] Hatcher, A. ; Press, Cambridge U. ; Mathematics, Cornell University. D.: *Algebraic Topology*. Cambridge University Press, 2002 (Algebraic Topology). `https://books.google.ch/books?id=BjKs86kosqgC`. – ISBN 9780521795401

[HSW+20] Hu, Yixin ; Schneider, Teseo ; Wang, Bolun ; Zorin, Denis ; Panozzo, Daniele: Fast tetrahedral meshing in the wild. In: *ACM Transactions on Graphics (TOG)* 39 (2020), Nr. 4, S. 117–1

[HTWB11] Huang, Jin ; Tong, Yiying ; Wei, Hongyu ; Bao, Hujun: Boundary aligned smooth 3D cross-frame field. In: *ACM transactions on graphics (TOG)* 30 (2011), Nr. 6, S. 1–8

[HXH10] Han, Shuchu ; Xia, Jiazhi ; He, Ying: Hexahedral shell mesh construction via volumetric polycube map. In: *Proceedings of the 14th ACM symposium on solid and physical modeling*, 2010, S. 127–136

[HZ16] Hu, Kangkang ; Zhang, Yongjie J.: Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation. In: *Computer Methods in Applied Mechanics and Engineering* 305 (2016), S. 405–421

[HZG+18a] Hu, Yixin ; Zhou, Qingnan ; Gao, Xifeng ; Jacobson, Alec ; Zorin, Denis ; Panozzo, Daniele: Tetrahedral meshing in the wild. In: *ACM Trans. Graph.* 37 (2018), Nr. 4, S. 60–1

[HZG+18b] Hu, Yixin ; Zhou, Qingnan ; Gao, Xifeng ; Jacobson, Alec ; Zorin, Denis ; Panozzo, Daniele: Tetrahedral Meshing in the Wild. In: *ACM Trans. Graph.* 37 (2018), Juli, Nr. 4, S. 60:1–60:14. – ISSN 0730–0301

[ISS09] Ito, Yasushi ; Shih, Alan M. ; Soni, Bharat K.: Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. In: *International Journal for Numerical Methods in Engineering* 77 (2009), Nr. 13, S. 1809–1833

[JFH+15] Jiang, Tengfei ; Fang, Xianzhong ; Huang, Jin ; Bao, Hujun ; Tong, Yiying ; Desbrun, Mathieu: Frame field generation through metric customization. In: *ACM Transactions on Graphics (TOG)* 34 (2015), Nr. 4, S. 1–11

[JGTR16] Johnen, A. ; Geuzaine, C. ; Toulorge, T. ; Remacle, J.-F.: Efficient Computation of the Minimum of Shape Quality Measures on Curvilinear Finite Elements. In: *Procedia Engineering* 163 (2016), S. 328–339. – ISSN 1877–7058. – 25th International Meshing Roundtable

[JHW+14] Jiang, Tengfei ; Huang, Jin ; Wang, Yuanzhen ; Tong, Yiying ; Bao, Hujun: Frame Field Singularity Correction for Automatic Hexahedralization. In: *IEEE Transactions on Visualization and Computer Graphics* 20 (2014), Nr. 8, S. 1189–1199

[JWR17] Johnen, Amaury ; Weill, J-C ; Remacle, J-F: Robust and efficient validation of the linear hexahedral element. In: *Procedia engineering* 203 (2017), S. 271–283

[KCPS13] Knöppel, Felix ; Crane, Keenan ; Pinkall, Ulrich ; Schröder, Peter: Globally optimal direction fields. In: *ACM Transactions on Graphics (ToG)* 32 (2013), Nr. 4, S. 1–10

[KLF14] Kowalski, N ; Ledoux, F ; Frey, P: Block-structured hexahedral meshes for CAD models using 3d frame fields. In: *Procedia Engineering* 82 (2014), S. 59–71

[KLF16] Kowalski, Nicolas ; Ledoux, Franck ; Frey, Pascal: Smoothness driven frame field generation for hexahedral meshing. In: *Computer-Aided Design* 72 (2016), S. 65–77

[KLSO12] Kowalski, Nicolas ; Ledoux, Franck ; Staten, Matthew L. ; Owen, Steve J.: Fun sheet matching: towards automatic block decomposition for hexahedral meshes. In: *Engineering with Computers* 28 (2012), Nr. 3, S. 241–253

[KMJ+19] Koch, Sebastian ; Matveev, Albert ; Jiang, Zhongshi ; Williams, Francis ; Artemov, Alexey ; Burnaev, Evgeny ; Alexa, Marc ; Zorin, Denis ; Panozzo, Daniele: ABC: A Big CAD Model Dataset For Geometric Deep Learning. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019

[KNP07a] Kälberer, Felix ; Nieser, Matthias ; Polthier, Konrad: Quadcover-surface parameterization using branched coverings. In: *Computer graphics forum* Bd. 26 Wiley Online Library, 2007, S. 375–384

[KNP07b] Kälberer, Felix ; Nieser, Matthias ; Polthier, Konrad: QuadCover - Surface Parameterization using Branched Coverings. In: *Computer Graphics Forum* 26 (2007), Nr. 3, 375-384. `http://dx.doi.org/https://doi.org/10.1111/j.1467-8659.2007.01060.x`. – DOI https://doi.org/10.1111/j.1467–8659.2007.01060.x

[Knu98] Knupp, Patrick M.: Next-Generation Sweep Tool: A Method for Generating All-Hex Meshes on Two-and-One-Half Dimensional Geometries. In: *IMR* Citeseer, 1998, S. 505–513

[Knu01] Knupp, Patrick M.: Hexahedral and tetrahedral mesh untangling. In: *Engineering with Computers* 17 (2001), Nr. 3, S. 261–268

[Knu03] Knupp, Patrick M.: A method for hexahedral mesh shape optimization. In: *International journal for numerical methods in engineering* 58 (2003), Nr. 2, S. 319–332

[Kop09] Kopriva, David A.: *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers.* Springer Science & Business Media, 2009

[KYKK19] Kang, Yunku ; Yoon, Seung-Hyun ; Kyung, Min-Ho ; Kim, Myung-Soo: Fast and robust computation of the Hausdorff distance between triangle mesh and quad mesh for near-zero cases. In: *Computers & Graphics* 81 (2019), S. 61–72

[LBK16] Lyon, Max ; Bommes, David ; Kobbelt, Leif: HexEx: Robust hexahedral mesh extraction. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 4, S. 1–11

[LBW00] Lai, Mingwu ; Benzley, Steven ; White, David: Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. In: *International Journal for Numerical Methods in Engineering* 49 (2000), Nr. 1-2, S. 261–275

[LG97] Liu, Shang-Sheng ; Gadh, Rajit: Automatic hexahedral mesh generation by recursive convex and swept volume decomposition. In: *6th international meshing roundtable, Sandia National Laboratories* Citeseer, 1997, S. 217–231

[LLX+12] Li, Yufei ; Liu, Yang ; Xu, Weiwei ; Wang, Wenping ; Guo, Baining: All-hex meshing using singularity-restricted field. In: *ACM Transactions on Graphics (TOG)* 31 (2012), Nr. 6, S. 1–11

[LMA95] Li, TS ; McKeag, RM ; Armstrong, CG: Hexahedral meshing using midpoint subdivision and integer programming. In: *Computer methods in applied mechanics and engineering* 124 (1995), Nr. 1-2, S. 171–193

[LMPS16] Livesu, Marco ; Muntoni, Alessandro ; Puppo, Enrico ; Scateni, Riccardo: Skeleton-driven adaptive hexahedral meshing of tubular shapes. In: *Computer Graphics Forum* Bd. 35 Wiley Online Library, 2016, S. 237–246

[LPC21] Livesu, Marco ; Pitzalis, Luca ; Cherchi, Gianmarco: Optimal dual schemes for adaptive grid based hexmeshing. In: *ACM Transactions on Graphics (TOG)* 41 (2021), Nr. 2, S. 1–14

[LPP+20] Livesu, Marco ; Pietroni, Nico ; Puppo, Enrico ; Sheffer, Alla ; Cignoni, Paolo: Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing. In: *ACM Transactions on Graphics (TOG)* 39 (2020), Nr. 4, S. 121–1

[LS10] Ledoux, Franck ; Shepherd, Jason: Topological and geometrical properties of hexahedral meshes. In: *Engineering with Computers* 26 (2010), Nr. 4, S. 419–432

[LSVT15] Livesu, Marco ; Sheffer, Alla ; Vining, Nicholas ; Tarini, Marco: Practical hex-mesh optimization via edge-cone rectification. In: *ACM Transactions on Graphics (TOG)* 34 (2015), Nr. 4, S. 1–11

[LVS+13] Livesu, Marco ; Vining, Nicholas ; Sheffer, Alla ; Gregson, James ; Scateni, Riccardo: Polycut: Monotone graph-cuts for polycube base-complex construction. In: *ACM Transactions on Graphics (TOG)* 32 (2013), Nr. 6, S. 1–12

[LZC+18] Liu, Heng ; Zhang, Paul ; Chien, Edward ; Solomon, Justin ; Bommes, David: Singularity-constrained octahedral fields for hexahedral meshing. In: *ACM Trans. Graph.* 37 (2018), Nr. 4, S. 93–1

[LZS+21] Li, Lingxiao ; Zhang, Paul ; Smirnov, Dmitriy ; Abulnaga, S M. ; Solomon, Justin: Interactive all-hex meshing via cuboid decomposition. In: *ACM Transactions on Graphics (TOG)* 40 (2021), Nr. 6, S. 1–17

[MÄ88] Mäntilä, M.: *An Introduction to Solid Modeling*. Computer Science Press, Maryland, 1988

[Mar09] Maréchal, Loïc: Advances in octree-based all-hexahedral mesh generation: handling sharp features. In: *Proceedings of the 18th international meshing roundtable*. Springer, 2009, S. 65–84

[Mar16] Maréchal, Loïc: All hexahedral boundary layers generation. In: *Procedia engineering* 163 (2016), S. 5–19

[MESB08] Merkley, Karl ; Ernst, Corey ; Shepherd, Jason F. ; Borden, Michael J.: Methods and applications of generalized sheet insertion for hexahedral meshing. In: *Proceedings of the 16th International Meshing Roundtable* Springer, 2008, S. 233–250

[Mit96] Mitchell, Scott A.: A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In: *Annual Symposium on Theoretical Aspects of Computer Science* Springer, 1996, S. 465–476

[MJL+21] Mölder, Felix ; Jablonski, Kim P. ; Letcher, Brice ; Hall, Michael B. ; Tomkins-Tinch, Christopher H. ; Sochat, Vanessa ; Forster, Jan ; Lee, Soohyun ; Twardziok, Sven O. ; Kanitz, Alexander u. a.: Sustainable data analysis with Snakemake. In: *F1000Research* 10 (2021)

[MPZ14] Myles, Ashish ; Pietroni, Nico ; Zorin, Denis: Robust Field-Aligned Global Parametrization. In: *ACM Trans. Graph.* 33 (2014), jul, Nr. 4. `http://dx.doi.org/10.1145/2601097.2601154`. – DOI 10.1145/2601097.2601154. – ISSN 0730–0301

[MPZS20] Marschner, Zoë ; Palmer, David ; Zhang, Paul ; Solomon, Justin: Hexahedral Mesh Repair via Sum-of-Squares Relaxation. In: *Computer Graphics Forum* Bd. 39 Wiley Online Library, 2020, S. 133–147

[MSCR07] Mishra, Nutan ; Sarvate, Dinesh G. ; Chisholm, Adrienne ; Raab, Jesse J.: A Note on Non-Regular Planar Graphs. (2007)

[MUF21] Millen, SLJ ; Ullah, Z ; Falzon, BG: On the importance of finite element mesh alignment along the fibre direction for modelling damage in fibre-reinforced polymer composite laminates. In: *Composite Structures* 278 (2021), S. 114694

[NRP11] Nieser, Matthias ; Reitebuch, Ulrich ; Polthier, Konrad: Cubecover–parameterization of 3d volumes. In: *Computer graphics forum* Bd. 30 Wiley Online Library, 2011, S. 1397–1406

[OSE17] Owen, Steven J. ; Shih, Ryan M. ; Ernst, Corey D.: A template-based approach for parallel hexahedral two-refinement. In: *Computer-Aided Design* 85 (2017), S. 34–52

[PA97] Price, Mark A. ; Armstrong, Cecil G.: Hexahedral mesh generation by medial surface subdivision: Part II. Solids with flat and concave edges. In: *International Journal for Numerical Methods in Engineering* 40 (1997), Nr. 1, S. 111–136

[PAS95] Price, Mark A. ; Armstrong, Cecil G. ; Sabin, MA: Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges. In: *International Journal for Numerical Methods in Engineering* 38 (1995), Nr. 19, S. 3335–3359

[PBS20] Palmer, David ; Bommes, David ; Solomon, Justin: Algebraic representations for volumetric frame fields. In: *ACM Transactions on Graphics (TOG)* 39 (2020), Nr. 2, S. 1–17

[PCS+22] Pietroni, Nico ; Campen, Marcel ; Sheffer, Alla ; Cherchi, Gianmarco ; Bommes, David ; Gao, Xifeng ; Scateni, Riccardo ; Ledoux, Franck ; Remacle, Jean-Francois ; Livesu, Marco: Hex-Mesh Generation and Processing: a Survey. In: *arXiv preprint arXiv:2202.12670* (2022)

[PJHHXCK22] Paul, Zhang ; Judy (Hsin-Hui), Chiang ; Xinyi (Cynthia), Fan ; Klara, Mundilova: LOCAL DECOMPOSITION OF HEXAHEDRAL SINGULAR NODES INTO SINGULAR CURVES. (2022)

[PLC+21] Pitzalis, Luca ; Livesu, Marco ; Cherchi, Gianmarco ; Gobbetti, Enrico ; Scateni, Riccardo: Generalized adaptive refinement for grid-based hexahedral meshing. In: *ACM Transactions on Graphics (TOG)* 40 (2021), Nr. 6, S. 1–13

[PMC+16] Palacios, Jonathan ; Ma, Chongyang ; Chen, Weikai ; Wei, Li-Yi ; Zhang, Eugene: Tensor field design in volumes. In: *SIGGRAPH ASIA 2016 Technical Briefs*. 2016, S. 1–4

[PRR+22] Protais, François ; Reberol, Maxence ; Ray, Nicolas ; Corman, Etienne ; Ledoux, Franck ; Sokolov, Dmitry: Robust quantization for polycube maps. In: *Computer-Aided Design* 150 (2022), S. 103321

[RCR19] Reberol, Maxence ; Chemin, Alexandre ; Remacle, Jean-François: Multiple approaches to frame field correction for CAD models. In: *arXiv preprint arXiv:1912.01248* (2019)

[RGRS11] Ruiz-Gironés, Eloi ; Roca, Xevi ; Sarrate, Josep: Using a computational domain and a three-stage node location procedure for multi-sweeping algorithms. In: *Advances in Engineering Software* 42 (2011), Nr. 9, S. 700–713

[RGRS12] Ruiz-Gironés, Eloi ; Roca, Xevi ; Sarrate, Josep: The receding front method applied to hexahedral mesh generation of exterior domains. In: *Engineering with computers* 28 (2012), Nr. 4, S. 391–408

[RGRS14] Ruiz-Gironés, Eloi ; Roca, Xevi ; Sarrate, Jose: Optimizing mesh distortion by hierarchical iteration relocation of the nodes on the CAD entities. In: *Procedia Engineering* 82 (2014), S. 101–113

[RPPSH17] Rabinovich, Michael ; Poranne, Roi ; Panozzo, Daniele ; Sorkine-Hornung, Olga: Scalable locally injective mappings. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1

[RRGS10] Roca, Xevi ; Ruiz-Gironés, Eloi ; Sarrate, Josep: Receding front method: a new approach applied to generate hexahedral meshes of outer domains. In: *Proceedings of the 19th International Meshing Roundtable*. Springer, 2010, S. 209–225

[RSL16] Ray, Nicolas ; Sokolov, Dmitry ; Lévy, Bruno: Practical 3D frame field generation. In: *ACM Transactions on Graphics (TOG)* 35 (2016), Nr. 6, S. 1–9

[RVLL08] Ray, Nicolas ; Vallet, Bruno ; Li, Wan C. ; Lévy, Bruno: N-symmetry direction field design. In: *ACM Transactions on Graphics (TOG)* 27 (2008), Nr. 2, S. 1–13

[SBB+22] Schmidt, Patrick ; Born, Janis ; Bommes, David ; Campen, Marcel ; Kobbelt, Leif: TinyAD: Automatic Differentiation in Geometry Processing Made Simple. In: *Computer graphics forum* Bd. 41 Wiley Online Library, 2022, S. 113–124

[SBO06] Scott, Michael A. ; Benzley, Steven E. ; Owen, Steven J.: Improved many-to-one sweeping. In: *International journal for numerical methods in engineering* 65 (2006), Nr. 3, S. 332–348

[Sch96] Schneiders, Robert: A grid-based algorithm for the generation of hexahedral element meshes. In: *Engineering with computers* 12 (1996), Nr. 3, S. 168–177

[SEK+07] Stimpson, C ; Ernst, CD ; Knupp, P ; Pébay, PP ; Thompson, D: The verdict library reference manual. In: *Sandia National Laboratories Technical Report* 9 (2007), Nr. 6

[SH77] Schmeichel, EF ; Hakimi, SL: On planar graphical degree sequences. In: *SIAM Journal on Applied Mathematics* 32 (1977), Nr. 3, S. 598–609

[SHG+22] Schneider, Teseo ; Hu, Yixin ; Gao, Xifeng ; Dumas, Jérémie ; Zorin, Denis ; Panozzo, Daniele: A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method. In: *ACM Transactions on Graphics (TOG)* 41 (2022), Nr. 3, S. 1–14

[SKO+10] Staten, Matthew L. ; Kerr, Robert A. ; Owen, Steven J. ; Blacker, Ted D. ; Stupazzini, Marco ; Shimada, Kenji: Unconstrained plastering—Hexahedral mesh generation via advancing-front geometry decomposition. In: *International*

journal for numerical methods in engineering 81 (2010), Nr. 2, S. 135–171

[SKOB06] Staten, Matthew L. ; Kerr, Robert A. ; Owen, Steven J. ; Blacker, Ted D.: Unconstrained paving and plastering: Progress update. In: *proceedings of the 15th International Meshing Roundtable* Springer, 2006, S. 469–486

[SLK04] Su, Yi ; Lee, KH ; Kumar, A S.: Automatic hexahedral mesh generation for multi-domain composite models using a hybrid projective grid-based method. In: *Computer-Aided Design* 36 (2004), Nr. 3, S. 203–215

[SOB05] Staten, Matthew L. ; Owen, Steven J. ; Blacker, Ted D.: Unconstrained paving & plastering: A new idea for all hexahedral mesh generation. In: *proceedings of the 14th International Meshing Roundtable* Springer, 2005, S. 399–416

[SR15] Sokolov, Dmitry ; Ray, Nicolas: *Fixing normal constraints for generation of polycubes*, LORIA, Diss., 2015

[SS96] Shih, Bih-Yaw ; Sakurai, Hiroshi: Automated hexahedral mesh generation by swept volume decomposition and recomposition. In: *5th International meshing roundtable* Bd. 280 Citeseer, 1996

[ST06] Si, Hang ; TetGen, A: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. In: *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany* 81 (2006)

[SVB17] Solomon, Justin ; Vaxman, Amir ; Bommes, David: Boundary element octahedral fields in volumes. In: *ACM Transactions on Graphics (TOG)* 36 (2017), Nr. 4, S. 1

[THCM04] Tarini, Marco ; Hormann, Kai ; Cignoni, Paolo ; Montani, Claudio: Polycube-maps. In: *ACM transactions on graphics (TOG)* 23 (2004), Nr. 3, S. 853–860

[ULP+15] Usai, Francesco ; Livesu, Marco ; Puppo, Enrico ; Tarini, Marco ; Scateni, Riccardo: Extraction of the quad layout of a triangle mesh guided by its curve skeleton. In: *ACM Transactions on Graphics (TOG)* 35 (2015), Nr. 1, S. 1–13

[VCD+16] Vaxman, Amir ; Campen, Marcel ; Diamanti, Olga ; Panozzo, Daniele ; Bommes, David ; Hildebrandt, Klaus ; Ben-Chen, Mirela: Directional field synthesis, design, and processing. In: *Computer graphics forum* Bd. 35 Wiley Online Library, 2016, S. 545–572

[VO19] Viertel, Ryan ; Osting, Braxton: An Approach to Quad Meshing Based on Harmonic Cross-Valued Maps and the Ginzburg–Landau Theory. In: *SIAM Journal on Scientific Computing* 41 (2019), Nr. 1, A452-A479. `http://dx.doi.org/10.1137/17M1142703`. – DOI 10.1137/17M1142703

[VPR19] Verhetsel, Kilian ; Pellerin, Jeanne ; Remacle, Jean-François: A 44-element mesh of Schneiders' pyramid: Bounding the difficulty of hex-meshing problems. In: *Computer-Aided Design* 116 (2019), S. 102735. – ISSN 0010–4485

[VSL16] Viertel, Ryan ; Staten, Matthew L. ; Ledoux, Franck: Analysis of Non-Meshable Automatically Generated Frame Fields. / Sandia National Lab.(SNL-NM), Albuquerque, NM (United States). 2016. – Forschungsbericht

[VTM+08] Vassberg, John C. ; Tinoco, Edward N. ; Mani, Mori ; Brodersen, Olaf P. ; Eisfeld, Bernhard ; Wahls, Richard A. ; Morrison, Joseph H. ; Zickuhr, Thomas ; Laflin, Kelly R. ; Mavriplis, Dimitri J.: Abridged summary of the third AIAA computational fluid dynamics drag prediction workshop. In: *Journal of Aircraft* 45 (2008), Nr. 3, S. 781–798

[Wat81] Watson, David F.: Computing the Delaunay Tesselation with Application to Voronoi Polytopes. In: *The Computer Journal* 24 (1981), S. 167–172

[WB06] Wächter, Andreas ; Biegler, Lorenz T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. In: *Mathematical programming* 106 (2006), Nr. 1, S. 25–57

[WBM21] Whalen, Eamon ; Beyene, Azariah ; Mueller, Caitlin: SimJEB: Simulated Jet Engine Bracket Dataset. In: *Computer Graphics Forum* (2021). – ISSN 1467–8659

[WGWC18] Wu, Haiyan ; Gao, Shuming ; Wang, Rui ; Chen, Jinming: Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing. In: *Computer-Aided Design* 96 (2018), S. 42–58

[WGZC16] Wang, Rui ; Gao, Shuming ; Zheng, Zhihao ; Chen, Jinming: Frame field guided topological improvement for hex mesh using sheet operations. In: *Procedia engineering* 163 (2016), S. 276–288

[WN+99] Wright, Stephen ; Nocedal, Jorge u. a.: Numerical optimization. In: *Springer Science* 35 (1999), Nr. 67-68, S. 7

[WPL+21] Willis, Karl D. ; Pu, Yewen ; Luo, Jieliang ; Chu, Hang ; Du, Tao ; Lambourne, Joseph G. ; Solar-Lezama, Armando ; Matusik, Wojciech: Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. In: *ACM Transactions on Graphics (TOG)* 40 (2021), Nr. 4, S. 1–24

[WSC+17] Wang, Rui ; Shen, Chun ; Chen, Jinming ; Wu, Haiyan ; Gao, Shuming: Sheet operation based block decomposition of solid models for hex meshing. In: *Computer-Aided Design* 85 (2017), S. 123–137

[WSK+15] Wu, Zhirong ; Song, Shuran ; Khosla, Aditya ; Yu, Fisher ; Zhang, Linguang ; Tang, Xiaoou ; Xiao, Jianxiong: 3d shapenets: A deep representation for volumetric shapes. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, S. 1912–1920

[WSRRR+12] Wilson, Thomas J. ; Sarrate Ramos, Josep ; Roca Ramón, Xavier ; Montenegro, R ; Escobar, JM: Untangling and smoothing of quadrilateral and hexahedral meshes. In: *Civil-Comp Proceedings* (2012)

[XGC18] Xu, Kaoji ; Gao, Xifeng ; Chen, Guoning: Hexahedral mesh quality improvement via edge-angle optimization. In: *Computers & Graphics* 70 (2018), S. 17–27

[XGDC17] Xu, Kaoji ; Gao, Xifeng ; Deng, Zhigang ; Chen, Guoning: Hexahedral meshing with varying element sizes. In: *Computer Graphics Forum* Bd. 36 Wiley Online Library, 2017, S. 540–553

[XLZ+21] Xu, Gang ; Ling, Ran ; Zhang, Yongjie J. ; Xiao, Zhoufang ; Ji, Zhongping ; Rabczuk, Timon: Singularity structure simplification of hexahedral meshes via weighted ranking. In: *Computer-Aided Design* 130 (2021), S. 102946

[YFL19] Yang, Yang ; Fu, Xiao-Ming ; Liu, Ligang: Computing Surface PolyCube-Maps by Constrained Voxelization. In: *Computer Graphics Forum* Bd. 38 Wiley Online Library, 2019, S. 299–309

[YSC21] Yu, Chris ; Schumacher, Henrik ; Crane, Keenan: Repulsive curves. In: *ACM Transactions on Graphics (TOG)* 40 (2021), Nr. 2, S. 1–21

[YWL+20] Yu, Yuxuan ; Wei, Xiaodong ; Li, Angran ; Liu, Jialei G. ; He, Jeffrey ; Zhang, Yongjie J.: Hexgen and hex2spline: polycube-based hexahedral mesh generation and spline modeling for isogeometric analysis applications in ls-dyna. In: *arXiv preprint arXiv:2011.14213* (2020)

[YZL15] Yu, Wuyi ; Zhang, Kang ; Li, Xin: Recent algorithms on automatic hexahedral mesh generation. In: *2015 10th International Conference on Computer Science & Education (ICCSE)* IEEE, 2015, S. 697–702

[ZB06] Zhang, Yongjie ; Bajaj, Chandrajit: Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. In: *Computer methods in applied mechanics and engineering* 195 (2006), Nr. 9-12, S. 942–960

[ZBG+07] Zhang, Yongjie ; Bazilevs, Yuri ; Goswami, Samrat ; Bajaj, Chandrajit L. ; Hughes, Thomas J.: Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow. In: *Computer methods in applied mechanics and engineering* 196 (2007), Nr. 29-30, S. 2943–2959

[ZJ16] Zhou, Qingnan ; Jacobson, Alec: Thingi10K: A Dataset of 10,000 3D-Printing Models. In: *arXiv preprint arXiv:1605.04797* (2016)

[ZLX13] Zhang, Yongjie ; Liang, Xinghua ; Xu, Guoliang: A robust 2-refinement algorithm in octree or rhombic dodecahedral tree based all-hexahedral mesh generation. In: *Computer Methods in Applied Mechanics and Engineering* 256 (2013), S. 88–100

[ZZM07] Zhang, Hongmei ; Zhao, Guoqun ; Ma, Xinwu: Adaptive generation of hexahedral element mesh using an improved grid-based method. In: *Computer-Aided Design* 39 (2007), Nr. 10, S. 914–928

Declaration of consent

on the basis of Article 18 of the PromR Phil.-nat. 19

Name/First Name: Liu Heng

Registration Number: 18-128-900

Study program: Computer Science

Bachelor ☐ Master ☐ Dissertation ☑

Title of the thesis: Frame Fields for Hexahedral Meshing

Supervisor: David Bommes

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of September 5th, 1996 and Article 69 of the University Statute of June 7th, 2011 is authorized to revoke the doctoral degree awarded on the basis of this thesis.
For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with theses submitted by others.

Bern, 09/03/2023

Place/Date

Signature Heng Liu