From Continuous Parametrizations to Discrete Hexahedral Meshes

Reliable Algorithms for a Versatile Pipeline

Hendrik Brückler

Doctoral Thesis

submitted in partial fulfillment of the requirements for the degree of
Doktor der Naturwissenschaften (Dr. rer. nat.)
at the Faculty of Computer Science, Electrical Engineering and Mathematics

Supervisor:
Prof. Dr. Marcel Campen

Paderborn, Germany
March 2026

Accepted upon recommendation of:

Prof. Dr. Marcel Campen (Paderborn University)
Dr. habil. Pierre Alliez (INRIA Sophia Antipolis)
Prof. Dr. Enrico Puppo (Università di Genova)

Defended:

February 25th, 2026

Abstract

Hexahedral meshes are the preferred volume discretization in many application domains, most importantly for physically-based simulations. The generation of such meshes for arbitrary input volumes, referred to as *hex meshing*, is a notoriously hard problem for which no fully satisfactory algorithmic solution exists to date.

Hex meshing methods based on parametrizations of the input volume offer the greatest potential in terms of input generality and output quality. In such methods the volume is mapped onto a dynamically determined 3D shape, on which a hexahedral mesh is trivially defined. The class of parametrization methods, due to their ambition of providing a truly general solution to hex meshing, is known to be particularly complex, so concrete instances of this class currently suffer from a variety of robustness issues that prevent their reliable application in practice.

This thesis develops various algorithms that serve as guaranteed-robust replacements for fragile solutions from prior literature. Together, they provide a reliable way of transforming continuous maps, so called *seamless maps*, into the discrete *integer-grid maps* eventually required for hex meshing, so that upstream pipelines can target this much simpler continuous type of maps.

The thesis is structured in three main parts. After establishing a general framework for representing meshes and parametrizations, the first main part of the thesis focuses on finding a coarse decomposition of a given input volume into cuboid-like sub-volumes. In analogy to concepts from surface meshing, these decompositions are formalized as *volume T-meshes*, and serve as both data structures and tools for the formulation of our subsequent robust algorithms.

In the second part, such T-mesh volume decompositions are used to replace the complex global problem—of transforming a seamless into an integer-grid map—by simpler but equivalent proxy problems, for which guaranteed-valid algorithmic solutions are described. The solutions of these proxy problems can then be composed back into a global solution. A key factor, contributing to the effectiveness of this approach, is that the algorithmic complexity is largely decoupled from the geometric complexity of the input volume and its representation; instead it depends only on the (much lower) complexity of the coarse T-mesh.

The final part then explores ways to further improve the efficiency, versatility and usability of the pipeline established above, while maintaining all robustness guarantees. In total, while not closing all existing robustness gaps of parametrization-based hexahedral meshing, the proposed algorithms vastly increase the success rate of state-of-the-art pipelines, as demonstrated throughout the thesis, while achieving at least similar and sometimes higher levels of result quality, with respect to results produced by prior fragile solutions—provided the latter produce valid output at all. To foster adoption by the community, potential applications of the proposed algorithms beyond the scope of this thesis are pointed out and open source reference implementations for all algorithms are released.

Zusammenfassung

Hexaedernetze (im Englischen: *hex meshes*) sind eine stark nachgefragte Repräsentationsform von digitalen volumetrischen Objekten, vor allem für Physik-Simulation. Solche Hexaedernetze zerteilen ein gegebenes Volumen in bündig miteinander abschließende, würfelartige Einheiten. Die Erzeugung solcher Netze für beliebige Eingabevolumina ist ein bekanntermaßen komplexes Problem, für das es bisher keine vollständig zufriedenstellende algorithmische Lösung gibt.

Unter der Vielzahl an Lösungsansätzen, haben solche, die auf *Volumen-Parametrisierung* beruhen, theoretisch das größte Potenzial beliebige Eingaben handhaben zu können und hochqualitative Netze zu erzeugen. Bei solchen Methoden wird das Eingabe-Volumen in eine dynamisch bestimmte 3D-Form transformiert wird, für die ein Hexaedernetz trivial gegeben ist. Die entsprechende Rücktransformation dieses Netzes erzeugt dann ein gültiges Netz im Original-Volumen. Allerdings sind Ansätze dieser Art aufgrund ihrer angestrebten Allgemeingültigkeit besonders komplex, und konkrete Instanzen weisen derzeit eine Vielzahl von Problemen auf, die ihre praktische Anwendbarkeit stark einschränken.

In dieser Arbeit werden Algorithmen entwickelt, die als garantiert zuverlässiger Ersatz für fehleranfällige Lösungen aus der bisherigen Literatur dienen. Zusammen ermöglichen sie gängigen Netzerzeugungs-Pipelines, eine wesentlich einfachere Art von Parametrisierungen, kontinuierliche *seamless maps*, anzustreben, indem sie eine zuverlässige Möglichkeit bieten, diese in die diskreten *integer-grid maps* zu überführen, die letztendlich zu Hexaedernetzen äquivalent sind.

Die Arbeit gliedert sich in drei Hauptteile. Nach der Formalisierung von Netzen und Parametrisierungen strebt der erste Hauptteil eine grobe Zerlegung eines gegebenen Eingabevolumens in quaderartige Teilvolumina an. Diese Zerlegungen werden als *T-Netze* formalisiert, und dienen als Datenstrukturen und Werkzeuge für die Formulierung der nachfolgenden robusten Algorithmen.

Im zweiten Teil werden solche T-Netze verwendet, um das komplexe globale Problem—die Umwandlung einer *seamless map* in eine *integer-grid map*—durch einfachere, aber äquivalente Stellvertreter-Probleme zu ersetzen. Die beschriebenen, garantiert gültigen Lösungen der einzelnen Stellvertreter-Probleme können dann wieder zu einer global gültigen Lösung zusammengesetzt werden. Entscheidend hierbei ist, dass die algorithmische Komplexität weitgehend von der geometrischen Komplexität des Eingabevolumens und seiner Darstellung entkoppelt wird; stattdessen hängt sie lediglich von der (viel geringeren) Komplexität des groben T-Netzes ab.

Im letzten Teil werden Möglichkeiten untersucht, die Vielseitigkeit und praktische Nutzbarkeit der zuvor etablierten Pipeline zu verbessern. Insgesamt schließen die entwickelten Algorithmen zwar nicht alle bestehenden Lücken parametrisierungsbasierter Ansätze, erhöhen jedoch die Erfolgsrate moderner Pipelines erheblich, wie über den Verlauf der Arbeit demonstriert wird. Dabei erzeugen sie Netze von ähnlicher bis höherer Qualität als frühere, fehleranfällige Lösungen—sofern letztere überhaupt gültige Ausgabe erzeugen. Um die Nutzbarkeit der vorgeschlagenen Methodik in Forschung und Anwendungen zu begünstigen, werden Anwendungsmöglichkeiten der vorgeschlagenen Algorithmen über den Rahmen dieser Arbeit hinaus aufgezeigt und Open-Source-Implementationen für alle involvierten Algorithmen veröffentlicht.

Acknowledgements

First and foremost, I want to express my gratitude to my advisor, Marcel, for his invaluable guidance and support throughout this journey. Especially in the first years of my PhD—when things were not going smoothly for me, both in terms of research and life beyond that—his patience, trust in my abilities, and genuinely active mentorship were a major reason I kept going. I am also very thankful to Pierre and Enrico for kindly agreeing to review this thesis, and to the DFG for funding parts of my research.

I am likewise particularly grateful to my lab colleagues—especially Steffen, Ingmar and Payam—who I really should call friends. Thank you all for fostering a truly wholesome research environment—shaped by interesting discussions and, perhaps even more importantly, by a shared willingness to goof around. Steffen, I want to thank you in particular for your TikZ wizardry, design counsel, and last-minute help in many projects.

Further, I want to thank my collaborators and acquaintances outside of the lab, and all the kind people I've come to know through academic visits, workshops and conferences. In particular I want to thank David, for many fruitful discussions, and for being such a supportive and genuinely nice person to work with.

Last but not least I want to thank my closer friends and family. I want to thank René and Franziska, for not letting spatial distance weaken our friendship over the years, for pushing our joint projects and for offering help in so many situations. Moritz, Artem, Alex, and Annika deserve special thanks as well—frequently spending time with you has truly enriched my life in Osnabrück. Magnus, thank you for being the older brother I will always look up to, but who has never seemed to look down on me, and for being there when I needed you the most. Gratitude is a word too insignificant for everything I owe to my parents, but of course I want to thank them too, for everything.

Finally, Kathrin—I trust that you know how much your presence, love, and support over all these years have meant, and continue to mean, to me.

Contents

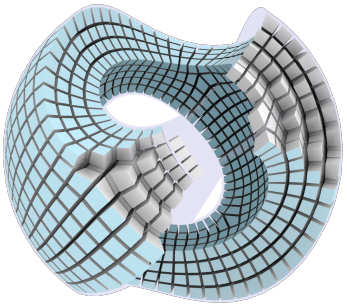
1 Introduction

Current times see an ever-increasing demand for digital 3D models across various application domains. In the entertainment industry, when working with actual physical objects is infeasible or too costly, creating a visually immersive experience requires high-quality digital copies of real-world entities. Depending on the artistic vision, the appearance of such models can reach from highly stylized to photorealistic.

At the core of 3D movies, games or visualization tools, these abstract models must be transformed into an *image* to display on a screen or similar device. In essence, this requires a simulation of real-world physical optics, typically called *rendering*. In this simulation a mathematical model for the interaction of light with matter is applied directly on the abstract data; the result is what a virtual camera would observe.

Visual appearance, however, is not the only physical property of interest in digital 3D models. In many application domains, including product design, medicine, engineering, and environmental studies, modeling realistic physical behavior in other aspects than optics is far more important. Furthermore, while mere plausibility of a model is sufficient for entertainment purposes, these other domains require *correctly* modeled behavior to succeed. The consequences of incorrectly predicted behavior typically encompass financial loss, but may reach far beyond that in safety-critical applications. The response of train wheels to mechanical stress [Esslinger et al. 2004], of a bridge to wind [Billah and Scanlan 1991], of a landform to torrential rain [Wartman et al. 2016], or of a fission reactor to power outage [Funabashi and Kitazawa 2012]—in these and many similar cases, the incorrect prediction of various physical phenomena has had disastrous consequences for individuals and society as a whole. A key to preventing such failures is to establish accurate digital models of the involved physical systems, and use them in preemptive simulations to predict their behavior under various conditions.

A high-accuracy physical simulation has two core requirements that are strongly coupled: First, a set of mathematical rules modeling the real-world physical process, and second, a matching representation of the spatial domain to which these rules are applied. In computer science terms, it requires simulation algorithms and spatial data structures to apply them to, with the choice of the former dictating the requirements on the latter. Hence, computationally obtaining accurate simulation results within a reasonable time hinges not only on algorithms but also on the format, accuracy and quality of their spatial input data. These three terms—format, accuracy and quality—should not be viewed as synonymous; each represents a distinct aspect of spatial data that independently affects the simulation outcome. The format indicates how the data is structured and encoded, the accuracy how closely it matches the real-world geometry, and the quality how well it satisfies numerical properties required by the simulation algorithms. Additionally, while optical behaviour of most objects is sufficiently captured via only their surface, most physical processes—or rather the mathematical models describing them—involve the interior of objects, and hence require a volumetric data representation.



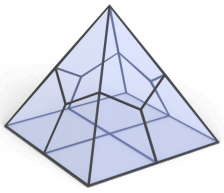
This thesis describes methods and tools for the generation of volumetric spatial data in a certain format, namely as so called hexahedral finite element meshes, in the following just called *hex meshes* for short. For a given shape, a hex mesh is conceptually obtained by gluing together small, potentially deformed cubes (referred to as hexahedra due to their six-sidedness), side-by-side to fill the shape's interior. This format has proven ideal for many simulation scenarios when compared to alternatives [Pietroni et al. 2022], but is also notoriously difficult to generate automatically, when high mesh accuracy and quality are required. Accuracy, in this case, indicates how closely the finite, discrete cubes match the object's shape along its surface, and potentially along special features such as sharp corners on the outside, or material boundaries on the inside. In contrast, quality refers to how similar the individual cubes are to perfect cubes (rectangular or even equilateral), i.e., typically the ideal case for numerical simulation.

1.1 Academic Interest

Beyond the perspective of obvious real-world applications, hex mesh generation is an intriguing research topic in its own right, combining various hard challenges from computer science and applied mathematics. Fundamentally, it revolves around the question: How can the highly-valued properties of a uniform cubic voxel grid (which trivially fills 3D space) be transferred to 3D volumes with complex boundaries without loss of detail? Despite the deceptive simplicity of this question, decades of research effort have not yielded a fully satisfactory solution to this problem as of yet [Pietroni et al. 2022].

While closely related to the well-studied *quad meshing* problem of adapting a 2D square grid to a given surface, the increase in dimensionality from 2D to 3D makes the problem significantly harder, and many concepts and tools from the 2D case do not carry over easily. More generally, it is well known that the complexity of many geometric problems increases drastically with their dimensionality [Werner 2013]. Indeed, many concrete problems that are comparatively easy to solve in 2D become much harder in 3D. Some that can be solved in linear or polynomial time in lower dimensions even become NP-hard in 3D [Knauer and Werner 2012; Mesmay et al. 2018].

Examples of problems closely related to hex meshing, for which straightforward solutions are available for surfaces but not for volumes, include the bijective convex mapping of surfaces [Tutte 1963] versus that of volumes [Meloni et al. 2024] and the splitting of convex polygons into quadrilaterals [Müller-Hannemann and Weihe 1997] versus that of convex polyhedra into hexahedra. The inset demonstrates an instance for the latter problem of finding a hexahedralization for a given boundary polyhedron, called Schneiders' pyramid [Schneiders 1996]. While seeming rather simple, it is surprisingly hard to solve and still treated in recent research [Verhetsel et al. 2019]. A recurring theme in this thesis is the generalization of 2D problems to their 3D counterparts, while drawing inspiration from the well-understood solutions of the 2D case to develop novel theory as well as algorithms for 3D.



1.2 Physical simulation

Not every simulation is based on 3D spatial data, not even simulations of pure-physics processes. A simulation of the behavior of an electric circuit, for instance, can be performed on the basis of a

spatially abstract network, involving only abstract nodes, links and attributes. On the other hand, physical processes that are inherently linked to 3D volumetric data include, but are not limited to:

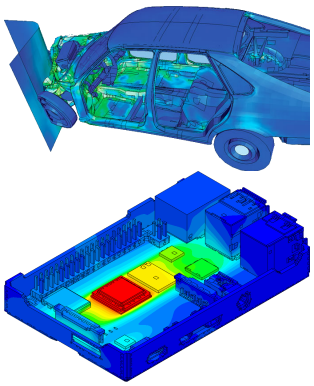
Structural mechanics The response of solid objects to external forces, in the form of deformation or fractures (inset, top).

Solid heat transfer The distribution of thermal energy in an object or fluid over time (inset, bottom).

Fluid dynamics The movement of liquids and gases in constrained spaces, such as air through a jet engine or water in a riverbed.

Electromagnetics The interaction between electromagnetic fields and matter, e.g., in *magnetic resonance imaging*.

Wave propagation The spreading of acoustic or electromagnetic waves through media.



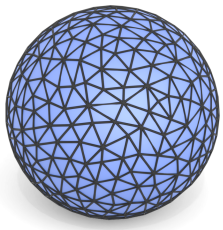
All these processes are governed by mathematical rules or, more concretely, by partial differential equations (PDEs). These can be used to predict the temporal behavior of a physical system, as they relate its change over time (and space) to its current (local) state. However, such differential equations for all but trivial scenarios do not have closed-form analytic solutions. So, even if a system's initial state could be expressed analytically, solving for its temporal evolution requires the application of numerical algorithms that can merely approximate the mathematical ground truth.

Finite element method The most common numerical procedure for approximate solving of PDEs, and also the one most relevant in the context of this thesis, is called the *finite element method* (FEM). One part of this approximation is to discretize both the continuous temporal and spatial domain; in other words, to reduce an infinite number of points in time and space to a finite one. This concretely means that time is divided into 1D steps, and space into 3D chunks of volume. These volume portions may be perfectly regular, such as the cells of a uniform cubic grid, or more irregular, such as a collection of various polyhedra glued together to fill a volume. The latter is essentially what is in the following called a *mesh*, or more precisely a *volume mesh*. Whenever the geometry of the spatial domain and its boundaries is complex—such that a grid would yield a poor approximation or an unwieldy number of cells—meshes are preferrable.

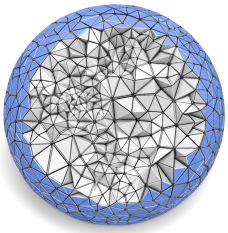
The remaining part of the approximation is to replace the continuous, global PDEs by discrete, piecewise algebraic equations that relate the relevant physical quantities, sampled at the chosen discrete points in time and space. For meshes, quantities are represented only at specific points within each cell, typically element corners, and related via equations only between neighboring points. Combining all these local equations results in a large but very sparse system of equations describing the entire system, which can efficiently be solved using numerical solvers.

1.3 Volume Meshes

Finite element meshes have been used over decades in various fields and industries, such as entertainment, mechanical engineering, environmental sciences, architecture, and urban modeling [Botsch et al. 2010]. In the fields of computer graphics and computer-aided design, the focus has traditionally been on visual appearance, requiring only surface representations. The simplest type of surface mesh, in terms of its elements, is one composed entirely of triangles, i.e., a triangle mesh (shown in the inset). Triangles, with three corners each,

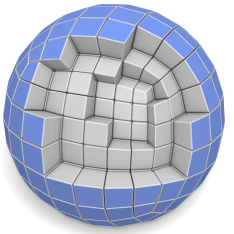


are the simplest possible polygons; yet a collection of triangles can approximate any surface, no matter how complex.



Going one dimension higher, the simplest type of volume mesh is one composed entirely of tetrahedra, i.e., a tetrahedral mesh (or *tet mesh* for short). A volume can likewise be approximated arbitrarily well, first approximating its boundary by a triangle mesh, and then filling its interior with tetrahedra (see the inset). Theory on the generation of both triangle and tet meshes is well-developed, and algorithms for creating them automatically, robustly and efficiently exist, even with quality guarantees (cf. section 2.2.4).

Meshes created in these ways, however, are referred to as *unstructured*, meaning that they have no regular arrangement of elements, with local element neighborhood structure varying greatly throughout the mesh. Consider, in contrast to that, a regular grid: In two dimensions, the regular square grid of the plane has equal connectivity at any grid point, meaning that every grid point has four direct neighbors. Furthermore, the alignment of cells in columns and rows implies far-ranging, predictable connections between elements. Lifting this 2D planar grid of quadrilaterals—with minor local connectivity adjustments—onto the surface of a 3D object creates what is sometimes called a structured quad mesh (shown in the inset). Notably, not every quad mesh is structured; for example, one obtained by simply splitting each triangle of a triangle mesh into three quads is not, as it is too dissimilar from a regular grid.



This concept extends to volume meshes: Filling a volume with cubes (or hexahedra, *hexes* for short) so that it resembles a regular cubic grid creates a structured hex mesh (inset), whereas simply splitting a tetrahedral mesh into a hex mesh would yield an unstructured one. Generally, we are interested in generating *structured hex meshes*; this term is more precisely disambiguated in section 2.3.3.2. The process of mesh generation is commonly referred to as *meshing*, and specifically *hex meshing* in our case.

To hex mesh or not to hex mesh?

While volume meshes consisting of general polyhedra have been used in the literature, the vast majority of research and infrastructure concerning volume meshes is based on either pure tetrahedral meshes or pure hexahedral meshes [Pietroni et al. 2022]. General feasibility of simulation is the same for either type; any physical process that can be simulated on one can also be simulated on the other. The difference lies in computational efficiency and accuracy of the simulation results. Historically, hexahedral meshes have been considered superior to tetrahedral ones in these regards, a belief that is in many cases well grounded in scientific data [Pietroni et al. 2022], but is seen by some scientists to be partially shaped by received wisdom, skewed comparisons and the presence of trusted legacy software that favors hex meshes [Erickson 2013; Schneider et al. 2022].

While this supposed superiority of hex meshes has sparked recent debate [Schneider et al. 2022], academia, industry and the open-source community are pushing for further advances in hex mesh generation [Pietroni et al. 2022], demonstrated by a growing number of scientific publications [Beaufort et al. 2021] and by the fact that hex meshes still are at the core of both commercial [Altair Inc 2025; ANSYS Inc 2025; COMSOL Inc 2025; Coreform Inc 2025; Dassault Systèmes Inc 2025] and open-source FEM software [Pommier and Renard 2005; Kirk et al. 2006; Alnæs et al. 2015; Schneider et al. 2019; Anderson et al. 2021; Arndt et al. 2023]. Section 2.3.5 provides a more detailed discussion of the benefits of hex meshes.

1.4 The Hex Meshing Problem

At its core, the problem of hex mesh generation has the following degrees of freedom (DOFs), and related objectives (analogous to similar 2D problems [Campen 2017]):

integer How many mesh elements (hexahedra, and their shared sides, edges and corners) to generate, so that element sizes locally match the feature size or a desired mesh resolution.

combinatorial How to connect the above elements, such that they yield a valid, well-structured mesh.

continuous How to embed the above elements into 3D space, such that input features are correctly fitted and each hexahedron is close to an ideal cube.

Because of this mixture of strongly coupled DOFs, constraints and objectives—of which a large number are discrete and thus hard to optimize for—no feasible all-in-one ('black-box') optimization solution to the problem is known so far.

Best-effort methods for the generation of hex meshes are discussed in various surveys. Just within these, the task of automatic and universally adaptable hex mesh generation has been described as "extremely difficult" [Shepherd and Johnson 2008], "the 'Holy Grail' of mesh generation research" [Blacker 2000], "an open problem" [Sarrate Ramos et al. 2014], and as "the most time-consuming task" in the overall FEM process [Tautges 2001; Sarrate Ramos et al. 2014]. The most recent among those surveys concludes [Pietroni et al. 2022]:

> *Despite the huge effort that various scientific and industrial communities have spent so far, the computation of a high-quality hexahedral mesh conforming to (or suitably approximating) a target geometry remains a challenge with various open aspects for which no fully satisfactory solutions have been provided yet. Some of the known methods are extremely robust and scale well on complex geometries; some others produce high-quality meshes; some others are fully automatic. But no known method successfully combines all these properties into a single product.*

Because the task of general, high-quality, automatic hex meshing continues to be an exceptionally hard challenge, even the most recent methods make compromises to simplify the problem in one or more of these aspects:

A Optimizing and fixing degrees of freedom sequentially rather than all at the same time, neglecting their interdependence, compromising quality.

B Shrinking the input space, e.g., to only tubular shapes, compromising generality.

C Shrinking the result space, e.g., by targeting only a subset of possible hex meshes, compromising generality and quality (as high-quality solutions might be cut off by this restriction).

D Requiring significant user (often expert) interaction, compromising speed and potentially all of the above.

E Widening the result space, e.g., by allowing non-hexahedral elements or non-exact feature alignment, compromising applicability.

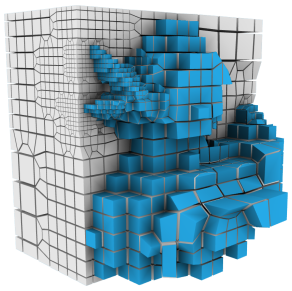
F Employing heuristic or numerically fragile routines that may (and often do) fail, compromising reliability.

These aspects are approximately ordered in how detrimental their effect is for the practical utility of a method, from least to most problematic.

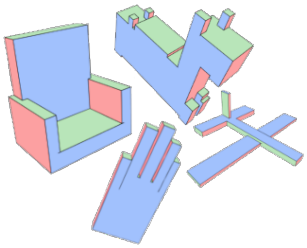
1.5 Common Hex Meshing Approaches

Virtually all hex meshing approaches employ **A** to some extent, although the incurred loss in optimality is usually secondary to other trade-offs. To illustrate the nature of such trade-offs, three common categories of approaches can be considered: Those based on grids, those based on polycube maps, and those based on integer-grid maps, the latter being the main focus of this thesis. Note that this is by no means an exhaustive classification of approaches, a more thorough overview is provided in section 2.3.7.

Grid-based approaches typically intersect the input volume to be meshed with a uniform or adaptive cubic grid, and include into the final mesh only those grid cubes that are fully covered by the volume. In a post-process, they then apply predefined templates to adapt the connectivity of the outer layer of cubes, and lastly attempt to fit this outer layer to the input volume's surface to minimize staircase artifacts. The approach, due to its conceptual simplicity, is extremely robust, but it both restricts the result space to meshes of limited quality (**C**, bad shapes are concentrated at the boundary), and does not stay within the set of admissible meshes, as it can not guarantee the exact recovery of input features (**E**).


(from: Livesu et al. [2021])

Approaches based on polycube maps, on the other hand, determine directional labels for each part of the input's surface (left, right, front, back, top, bottom), and then attempts to find a deformed version of the input volume that is exactly aligned to a cubic grid, with each surface part aligning with a grid layer of the prescribed orientation. This deformation can then be reversed and applied to the so covered portions of the cubic grid, pulling the cubes back into the original volume. The relation between the original and deformed-onto-grid version of the input volume is called a *polycube map*. If this map is *valid*, its inverse correctly deforms the grid cubes back into the input volume. However, there are currently no guaranteed ways to obtain a valid, general polycube map, only heuristic approaches (**F**). In practice, the success rate is still relatively high. The approach still shrinks the result space (**C**) and still has some problems preserving non-sharp features (**E**), but in both aspects much less so than the grid-based approach.

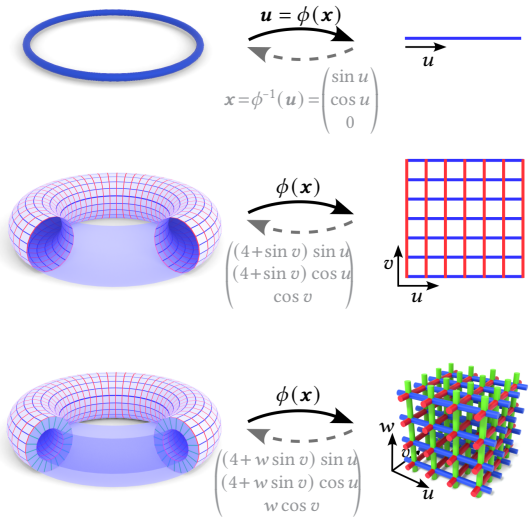

(from: Mandad et al. [2022])

The final family of approaches, those based on integer-grid maps, follow a similar idea, but additionally allow for arbitrary (partial and full) cuts through the volume before deforming it onto a grid—thereby exactly covering the complete range of hex meshes—and can guarantee exact preservation of input features. Due to these properties, they are considered to have the greatest potential in terms of generality and quality among hex meshing methods. However, existing methods in this category currently exhibit relatively high failure rates, because only fragile solutions have existed for several steps (**F**), including the (implicit) choice of cuts and the deformation computation.

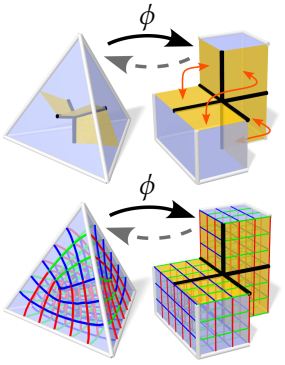
In this thesis, guaranteed reliable replacements for some of these fragile steps in the integer-grid map approach to hex meshing are devised, and the gains in robustness are demonstrated (mitigating **F**), while maintaining the intrinsic versatility and quality properties of the approach. Furthermore, improvements in quality (with respect to the effect of **A**) as well as in efficiency are implemented on top of the aforementioned reliable replacements. In this way, the presented contributions represent a major step forward in solving the open problem of general, high-quality, automatic, and reliable hex meshing.

1.5.1 Hex Meshing via Integer-Grid Parametrization

Parametrizations in our setting are essentially certain classes of mathematical maps $\phi : \mathbb{R}^3 \to \mathbb{R}^3$, i.e., ones that continuously map each point $\boldsymbol{x}$ in the input volume to a point $\boldsymbol{u}$ in some subset of another 3D space—and vice versa. We differentiate between the object space—the map's *domain*, which the input volume lives in (left in the inset)—and the parameter space—the codomain, where the input's image under the map lives in (right in the inset). The maps then create a local one-to-one correspondence between points in object and parameter space. An inverse view of such a map is that it models the input object as a function of three parameters. Hence maps of similar type are often referred to as a *parametrizations* of the input domain; we use these terms synonymously. With a cartesian coordinate system in parameter space, the *integer-grid* is exactly the grid formed by the points for which some parametric coordinates are integers. This grid exhibits cubic grid cells bounded by 8 grid corners each (bottom right in the inset). If the map $\phi(\boldsymbol{x})$ is chosen in such a way, that the volume's image in parameter space lines up exactly with this integer-grid, the point correspondence can be used via $\phi^{-1}(\boldsymbol{u})$ to transport the cubic integer-grid cells from parameter space back into object space, so that they exactly fill the input volume.



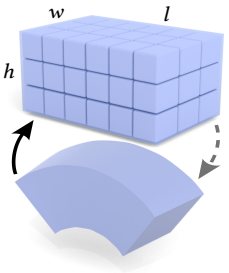
To this end, the volume's boundary and other critical elements are explicitly required to lie on integer coordinates in parameter space. This is displayed in the bottom of the inset. A map ϕ that satisfies these requirements is called an *integer-grid map*, but with a caveat: To cover the entire space of possible hex meshes, integer-grid maps allow cutting the input before mapping (orange in the inset). The resulting sides of the cut are then related via constraints, so that their image in parameter space overlays the integer grid in an equivalent way (orange arrows). Under this condition, the pulled back grid on the original input (bottom left in the inset) is continuous.



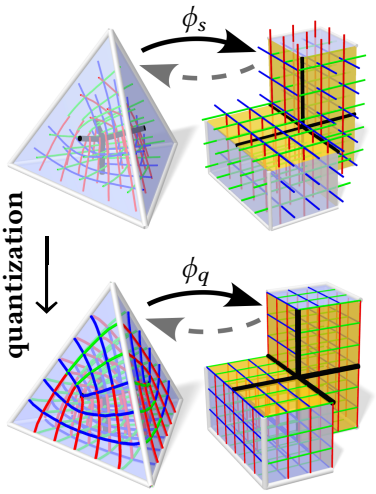
One advantage of the map-based approach is that the positioning of hex elements within the input can be tuned via incremental changes to the integer-grid map. Through this coupling, the resulting hex element quality can be improved by lowering the distortion of the map, which is roughly equivalent to the amount of local deformation between the input volume and its image in parameter space. Minimization of map distortion has always been a central topic in computer graphics and geometry processing, hence a vast amount of theoretical and practical solutions to this problem exist, readily usable to optimize for high quality in the generated meshes. This fully covers the continuous degrees of freedom (DOFs) of hex meshing.

Even more crucially—concerning the discrete DOFs—using maps for the purpose of hex meshing allows to sidestep (A) explicitly enumerating hex corners, and (B) explicitly connecting them, in favor of a representation that implicitly encodes both [Pietroni et al. 2022]. This reformulation drastically reduces the number of discrete variables and constraints, thereby bringing the problem within reach of practical solution strategies. To illustrate this, consider the image of a simple shape to be a cuboid block in parameter space, with a certain integer width w, height h and

length l, as shown in the inset. This block would then exactly cover $w \times h \times l$ grid cells and $(w+1) \times (h+1) \times (l+1)$ grid points. In this formalism, the implicit description via only three integers determines both integer DOFs (element count) and combinatorial DOFs (element connectivity) of the implied hex mesh. To express the same configuration, an explicit approach would have to individually handle $(w+1) \times (h+1) \times (l+1)$ discrete hex corners and their combinatorial connectivity.



As an additional advantage, the above formulation provides a straightforward and useful problem relaxation, namely allowing block sizes l, w, h to vary continuously rather than constraining them to integers. The core of this thesis is concerned with a practical and robust way of transforming the solution to such a continuous relaxation of the general integer-grid problem (a so called *seamless map* ϕ_s) into one that is a *quantized* seamless map ϕ_q, i.e., an integer-grid map. This process, demonstrated in the inset, is called *quantization*. Availability of a robust quantization makes solving the overall problem of map-based hex meshing significantly simpler, because earlier steps in the pipeline can then generate a *continuous* seamless map rather than a (partially) discrete integer-grid map, which is far easier.



1.5.2 Open Problems

Hex meshing based on integer-grid maps so far has proven to be a problem still too complex to solve in a single optimization step. Consequently, state-of-the-art pipelines consist of several substeps, each fixing only some of the problem's DOFs:

I Tet meshing of the domain.

II Computation of an orientation field on the tet mesh, determining how cubes of the later mesh should be oriented. This implies all necessary cuts through the domain.

III Computation of a seamless map on the tet mesh with the same cut structure.

IV Quantization of the seamless map into an integer-grid map, enforcing alignment of critical points to the integer grid.

V Hexahedral mesh extraction by pullback of the integer grid along the integer-grid map.

Among the steps listed above, the ones with dotted border are those for which only fragile solutions have existed so far. For step **II**, the success rate was significantly improved by advances in recent years [Liu and Bommes 2023]. However, the core problem of which orientation fields admit a valid hex mesh—and how to restrict field design to such meshable fields—is still largely unsolved.

The parametrization problem in step **III** is a highly non-convex constrained optimization problem. Even recent numerical optimization algorithms, albeit used with some success in practice, so far cannot guarantee that the resulting map is valid, which down the line often causes defects in the final hex mesh. However, robust combinatorial solutions to similar problems, albeit of simpler type, have been discovered recently [Campen et al. 2016; Hinderink and Campen 2023;

Nigolian et al. 2023; Hinderink et al. 2024; Nigolian et al. 2024], and it is conceivable that a robust extension to the problem type III will be found in the near future. Even then, such robust methods currently come with practical limitations, like high map distortion and steep increase in mesh complexity, which likewise need to be addressed in the future.

Interestingly, the complexity of the quantization step IV, i.e., of transforming a continuously relaxed seamless map into an actual integer-grid map, has been mostly disregarded in prior literature, and solutions have been employed for which failures commonly occur. This thesis specifically tackles previous robustness gaps of the quantization stage and provides reliable solutions that yield results at least equivalent in quality to the results of previous fragile methods—provided the latter produce valid output at all.

1.6 Thesis Structure and Contributions

Part A: Preliminaries provides a more detailed overview of concepts, notations and previous works relevant for the remainder of this thesis. Chapter 2 discusses types of volume meshes, the aspects of mesh structure, geometry, as well as approaches for mesh generation and trade-offs between these. Chapter 3 covers integer-grid maps in detail, including their properties, construction and use in hexahedral meshing.

Part B: T-meshes as Proxy Domains presents algorithms for partitioning a volumetric domain equipped with a seamless map into cuboid blocks. A first robustness gap, caused by the common presence of numerical errors in input seamless maps, is closed in chapter 4. Building on seamless maps numerically sanitized in this way, chapter 5 establishes a reliable method of constructing coarse cuboid-only partitions, serving as coarse proxy domains that enable efficient and provably correct solutions to the quantization problem. A clean formalization of such partitions as volumetric T-meshes is provided in chapter 6.

Part C: Robust Volume Quantization presents fully automatic methods for reliably quantizing a given seamless map into an integer-grid map. This part closes previously existing robustness gaps in step IV of the integer-grid map hex meshing pipeline. Chapter 7 demonstrates how T-meshes can be used in a proxy quantization step to determine exactly the integer DOFs of the integer-grid map problem. Chapter 8 then demonstrates how the so chosen integers can be transported down to the underyling map in a straightforward way, subsequently requiring only solving a fully continuous parametrization problem to obtain the final integer-grid map. Crucially, the integers fixed in this way are guaranteed to admit a valid integer-grid map, closing the first robustness gap. Finally, Chapter 9 explicates how the T-mesh can furthermore be used to solve the remaining continuous problem in a fully robust way, using the T-mesh block structure to build a complex map as a union of simple maps. This closes the final robustness gap of quantization.

Part D: Towards Versatility and Efficiency includes two augmentations of the above robust quantization procedure. First, chapter 10 covers how additional DOFs concerning the combinatorial structure of the integer-grid map, which are commonly assumed fixed from an early pipeline step onward, can be reconfigured during quantization. This opens up high-quality solutions precluded in earlier works. Second, chapter 11 details how employing black-box branch-and-bound solvers with problematic worst case behavior in the quantization step can be avoided, by devising a specialized, more efficient greedy algorithm that still achieves near-optimality on average.

Part E: Synopsis summarizes the contributions of this thesis, discusses the resulting pipeline's limitations, and outlines possible directions for future research.

Publications

The contents of this thesis are largely based on the following prepublications, of which the author of this thesis is the main author.

[Brückler et al. 2022b] H. Brückler, O. Gupta, M. Mandad, and M. Campen. 2022b. "The 3D Motorcycle Complex for Structured Volume Decomposition." *Computer Graphics Forum*, 41, 2

This publication received a *Best Paper (Honorable Mention)* award at Eurographics 2022. It relates to chapters 4 and 5, as well as the use cases outlined in chapter 6. Notable extensions presented in these chapters with respect to the original publication include: (i) a unified notation, (ii) a more thorough discussion of the sanitization routine, (iii) a rearranged proof for possible cell types of the motorcycle complex, and (iv) explicit support of features, originally introduced in a later publication.

[Brückler et al. 2022a] H. Brückler, D. Bommes, and M. Campen. 2022a. "Volume Parametrization Quantization for Hexahedral Meshing." *ACM Transactions on Graphics*, 41, 4

This publication, presented at SIGGRAPH 2022, is the basis for chapters 7 and 8. Notable extensions include: (i) a unified notation, (ii) a more thorough discussion of the sufficiency of separation constraints, (iii) an extended discussion of practical aspects of the reparametrization procedure, and (iv) explicit support of features, originally introduced in a later publication.

[Brückler and Campen 2023] H. Brückler and M. Campen. 2023. "Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing." *ACM Transactions on Graphics*, 42, 6

This publication, presented at SIGGRAPH Asia 2023, relates to chapter 9, as well as most of the formalisms introduced in chapter 6; a unified notation is established and comments on implementation-related aspects are added.

[Brückler et al. 2024] H. Brückler, D. Bommes, and M. Campen. 2024. "Integer-Sheet-Pump Quantization for Hexahedral Meshing." *Computer Graphics Forum*, 43, 5

This publication received a *Best Paper (Honorable Mention)* award at the Symposium on Geometry Processing 2024. It is the foundation for chapter 11; a unified notation is established and proper support for the feature-based objective function from a later publication is added.

[Brückler and Campen 2026] H. Brückler and M. Campen. 2026. "Volume Quantization with Flexible Singularities for Hexahedral Meshing." *Computer Graphics Forum*, 45, 2

This publication will be presented at Eurographics 2026 and covers chapter 10.

The contributions of all involved authors are declared in chapter I.

Part A

Preliminaries

I hate meshes.
I cannot believe how hard this is.
Geometry is hard.

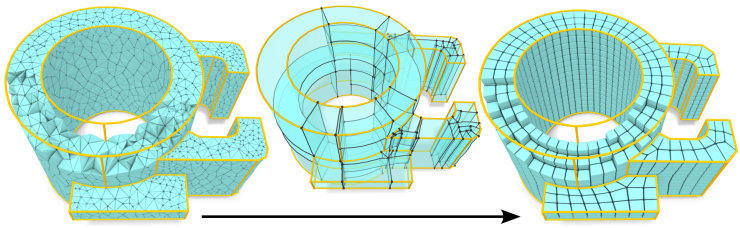
David Baraff
Senior Research Scientist
Pixar Animation Studios

Volume Meshes 2

I *Tet mesh* the input domain.

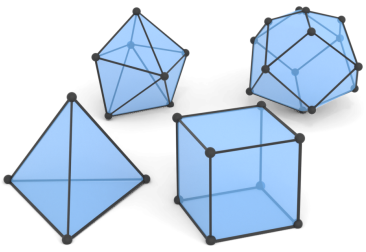
… …embed a T-mesh in it…

V Extract a *hex mesh*.



From a mathematical viewpoint, the combinatorial structure of meshes allows the reduction of otherwise continuous problems to (partially) combinatorial ones, which often admit easier theoretical analysis and more robust practical algorithms. In this thesis specifically, meshes are ubiquitous, appearing as inputs (tet meshes), intermediate domains (T-meshes), and target outputs (hex meshes). Therefore, the first goal is to define a unified description of volume meshes that can cover all these different variants in a consistent manner. The notation established here is concerned with the more standard case of *polyhedral meshes*, covering tetrahedral and hexahedral meshes as special cases, and serving as a baseline for the introduction of T-meshes in later chapters. Then, having established an intuition and formal language for discussing meshes, we review the aspects of tetrahedral and hexahedral meshes that are immediately relevant for their algorithmic processing and generation in the scope of this thesis.

2.1 Polyhedral Cell Complex



Polyhedra are three-dimensional solid bodies bounded by polygonal facets (blue in the inset), straight edges (black lines) and point-like vertices (black spheres). In geometrically strict definitions, the polygonal facets are often restricted to be flat; for triangular facets (left in the inset) this is always the case. However, for meshes involving quadrilateral facets bounded by four edges and vertices (right in the inset), this notion is too restrictive to allow meshes of practical relevance and is typically dropped. Our main interest are the simplest polyhedra achievable with triangular or quadrilateral facets, respectively, namely tetrahedra and cubic hexahedra (bottom row in the inset). While there are other polyhedra with six sides, we use the term hexahedra exclusively for those that are combinatorially equivalent to cubes, i.e., have six quadrilateral facets, twelve edges and eight vertices.

Moreover, we want to treat such polyhedra not in isolation, but as part of a larger structure: a *polyhedral mesh*. A high-level definition of polyhedral meshes is that they represent a decomposition of a volume into polyhedra—potentially from a restricted set—such that:

- The polyhedra fill the target volume without gaps.
- Adjacent polyhedra overlap only along their boundaries, meaning their interiors are disjoint.
- Adjacent polyhedra share a full facet, edge, or vertex.

While this is a geometric, intuitive way of defining a mesh, it does not explicitly capture its combinatorial structure. Thus, it also does not easily map to practical data structures commonly used to represent hex meshes. Hence, this thesis employs a more constructive definition which—while capturing the above idea—is more verbose but also offers more notational tools to work with meshes and analyze them.

2.1.1 Cell Complexes

Cell complexes are an abstraction commonly used to model and study the topology of mathematical spaces. General graphs (meaning networks here) can be considered cell complexes involving only 0-dimensional cells (vertices) and 1-dimensional cells (edges). Crucially, edges are defined not geometrically but combinatorially, i.e., one specifies which uniquely identified vertex connects to which edges and vice-versa. A planar graph or a surface mesh additionally contains 2-dimensional facets linked to the lower-dimensional cells. In our setting, the polyhedra in a polyhedral mesh are the 3-dimensional cells of a cell complex.

The literature on cell complexes is vast and dates back to pioneering works by Johann Benedict Listing, published as early as 1862 [Klette 2000]. Still, cell complexes as *discrete* models of topology are becoming increasingly relevant due to the development of digital technology, which is inherently discrete as well. In a survey by Klette [2000], the author discusses different paradigms for defining cell complexes employed throughout history. A distinction is made between more restricted, concrete definitions, in which a cell complex is defined via cells with a fixed geometric interpretation (e.g., a cell is an n-ball and such n-balls may overlap to create incidence relations), and more abstract, axiomatic definitions, in which cells remain abstract elements and only their incidences are defined. The widely used CW-complex formalism [Hatcher 2002, §2.1] is classified as somewhat of a middle ground, combining axiomatic definition with fixed geometric interpretation. The concept behind it, is to define k-dimensional skeletons X_k for a topological space, each formed by topological k-balls. These are then joined formally via continuous gluing maps g that "glue" the bounding k-spheres of each k-ball onto elements of the lower dimensional skeleta.

Indeed, on a theoretical level, a CW-complex is universal enough to model virtually all practical use cases and—along with the slightly more restricted Δ-complex variant—is widely used as a model for describing meshes that are more general than the most commonly used linear simplicial meshes (see e.g., Liu et al. [2018] and Sharp et al. [2019]). However, the CW-complex notation is rather unintuitive and does not translate easily to implementations in software.

For this reason, our proposed notation combines several concepts: We use the (more general) notation of abstract complexes (abstract elements and incidences) as defined by Klette [2000], but impose restrictions on the geometry of cells and the complex a posteriori. Formulating such restrictions after the fact is more verbose, and perhaps less elegant than postulating them via axioms, but has a major advantage: Restrictions can be (temporarily) dropped without the entire formalism being invalidated as a result. This is crucial for software implementations of modifiable meshes, which often require a space of possible configurations to traverse during modification, that is larger than the default space of (geometrically) valid configurations.

The *a posteriori* restrictions still allow to confirm that a given *non-temporary* mesh is valid (as per the use-case dependent definition of *valid*). Hence, the notation chosen here maps in

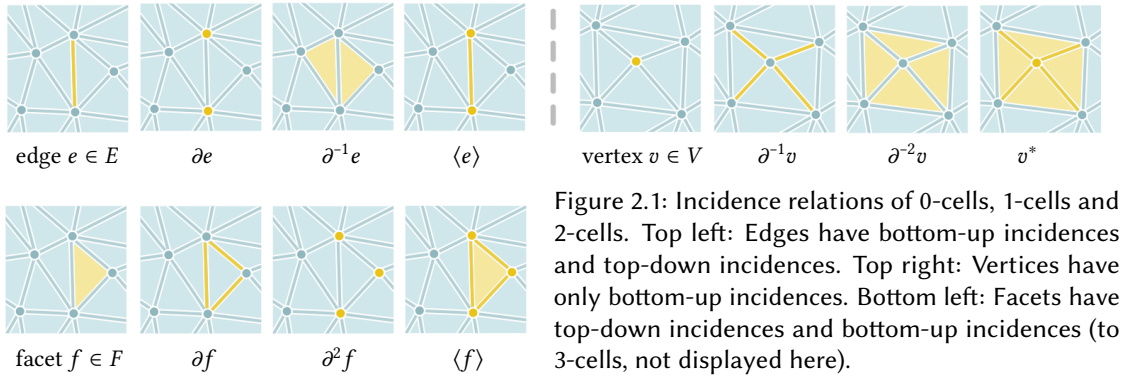


Figure 2.1: Incidence relations of 0-cells, 1-cells and 2-cells. Top left: Edges have bottom-up incidences and top-down incidences. Top right: Vertices have only bottom-up incidences. Bottom left: Facets have top-down incidences and bottom-up incidences (to 3-cells, not displayed here).

a straightforward manner to software implementations of meshes, like the OPENVOLUMEMESH datastructure [Kremer et al. 2013], which is also used as the underpinning in the code released alongside this work. Secondly, modular restrictions also allow to model various kinds of cell complexes used throughout this thesis with a unified formalism. This includes the concrete polyhedral meshes introduced in the following, but also the cell complexes representing high-level volume partitions, introduced in chapter 5, and others appearing as transitional representations in later chapters. The notational tools forming the basis of this unified formalism for describing cell complexes are introduced in the following.

2.1.2 Notation Tools

At the core of a cell complex lie its cells—quite literally. Just like a graph can be seen as a set of 1D vertices and 2D edges "glued" together in a sound way, a 3D cell complex can be assembled from a set of n-dimensional cells (n-cells) for $n \in \{0, 1, 2, 3\}$ and some "glue." These n-cells are abstract representations of entities, potentially with some geometric interpretation. In our context, n-cells represent open topological n-balls, i.e., entities that are locally homeomorphic to $\mathbb{R}^n$. In later chapters, this notion is temporarily broadened where necessary. The "glue" metaphor is realized via incidence operators that model the (topological) connectedness of cells. A 2D pictorial summary of the notation introduced below is shown in fig. 2.1.

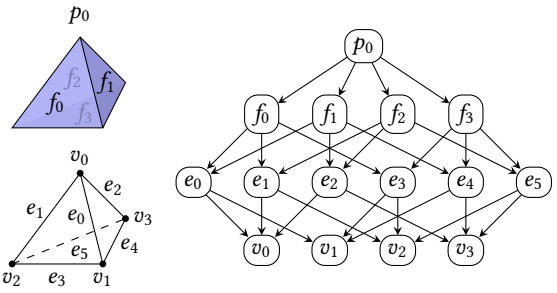
Notation 2.1 (n-cells). A volumetric cell complex is formed by the union of all its n-cells C_n with $n \in \{0, 1, 2, 3\}$: $C = C_3 \cup C_2 \cup C_1 \cup C_0$. In more descriptive terms, we denote 3-cells as **p**olyhedra $C_3 := P = \{p_1, p_2, \dots\}$, 2-cells as **f**acets $C_2 := F = \{f_1, f_2, \dots\}$, 1-cells as **e**dges $C_1 := E = \{e_1, e_2, \dots\}$, and 0-cells as **v**ertices $C_0 := V = \{v_1, v_2, \dots\}$. For any n-cell $c \in C_n$, $\dim(c) = n$ is called the *dimension* of a cell.

Notation 2.2 (Top-down incidences). Let ∂ be a top-down incidence operator that maps an n-cell to the set of its $(n-1)$-dimensional bounds.

In a concrete geometric interpretation, this means mapping an n-ball to the $(n-1)$-dimensional elements lying on the n-sphere around it. This operator alone is sufficient to define the gluing, but for convenience some syntactic sugar is introduced. The application of any operator introduced below to entire sets of cells $C' \in C$ is to be understood as the union over element-wise applications. Note that these incidences, due to only being defined between cells differing in dimension by exactly 1, preclude some conceivable—but hardly practically relevant—configurations, where an n-cell would be directly glued to an $(n-2)$-cell. As an example of such, imagine for instance a

disk-like facet, where all points on the disk boundary are identified as a single vertex, creating a kind of "balloon." Such configurations are not required in the context of this thesis, so their representation can be safely discarded.

Notation 2.3 (Incidence graphs). The incidence relation ∂ induces a partial ordering of cells in a complex. Hence, it is possible to model incidences via hierarchical trees, i.e., directed acyclic graphs, with each edge between c and c' corresponding to a single relation $c' \in \partial c$. The inset demonstrates this for a single tetrahedron.



The graph can be interpreted as a set of gluing instructions; as evident from the inset, these instructions exhibit a high combinatorial complexity for even the simplest examples.

Notation 2.4 (Derived incidences). Let ∂^{-1} be a bottom-up incidence operator defined as the inverse of ∂ via $s \in \partial^{-1} s' \iff s' \in \partial s$. For an n-cell $c \in C$ let furthermore $\partial^2 c = \bigcup_{c' \in \partial c} \partial c'$ and $\partial^3 c = \bigcup_{c' \in \partial^2 c} \partial c'$, i.e., ∂^k yields the set of transitive $(n-k)$-dimensional bounds. Let ∂^{-k} be defined equivalently to yield the set of transitively bounded $(n+k)$-dimensional cells of an n-cell.

Notation 2.5 (Closure and star). For any cell $c \in C$ its closure $\langle c \rangle$ is defined as the union of c with all cells that (transitively) bound it: $\langle c \rangle = \{c\} \cup \bigcup_k \partial^k c$. The star c^* is defined as the union of c with all cells that are (transitively) bounded by c, i.e., $c^* = c \cup \bigcup_k \partial^{-k} c$.

Wording For any two cells $c, c' \in C$ iff $c \neq c'$ and $c' \in \langle c \rangle$ we say c' *bounds* c or, equivalently, c is bounded by c'. If either c bounds c' or vice-versa, we say c and c' are *incident* (to one another). Two cells of the same dimension $c \neq c' \in C_n$ cannot be incident; if, however, $\langle c \rangle \cap \langle c' \rangle = C' \neq \emptyset$, we call them *adjacent*. If additionally, C' contains one or more $(n-1)$-cells, we call them *directly adjacent*.

2.1.3 Definitions

Definition 2.6 (Abstract polyhedral mesh). A volumetric cell complex $\mathcal{P} = P \cup F \cup E \cup V$ equipped with incidences ∂ is called an abstract polyhedral mesh if:

($\mathcal{P}_\text{I}$) The mesh is finite: $|\mathcal{P}| \in \mathbb{Z}$

($\mathcal{P}_\text{II}$) The bounds of each cell are also part of the mesh:

$$\forall s \in \mathcal{P} : \partial s \subseteq \mathcal{P}.$$

($\mathcal{P}_\text{III}$) n-cells are open n-balls glued to boundary elements forming a closed n-sphere:

(a) $\forall v \in V : \partial v = \emptyset$, i.e., vertices are not bounded.

(b) $\forall e \in E : |\partial e| = 2$, i.e., edges are bounded by two distinct vertices.

(c) $\forall f \in F : |\partial f| = \left|\partial^2 f\right| \geq 3$, i.e., facets are bounded by three or more edges and equally many vertices.

(d) $\forall f \in F : \forall v \in \partial^2 f : |\partial^{-1} v \cap \partial f| = 2$, i.e., facets are bounded in a manifold fashion.

(e) $\forall f \in F : \forall E_f \subsetneq \partial f : \left(E_f = \emptyset \vee \exists v \in \partial E_f : |\partial^{-1} v \cap E_f| \neq 2\right)$, i.e., facets are bounded by exactly one closed loop.

(f) $\forall p \in P : |\partial p| \geq 4$, i.e., polyhedra are bounded by four or more facets.

(g) $\forall p \in P : \forall e \in \partial^2 p : |\partial^{-1} e \cap \partial p| = 2$, i.e., polyhedra are bounded in an edge-manifold fashion.

(h) $\forall p \in P : \forall v \in \partial^3 p : \forall F_v \subsetneq \partial^{-2} v \cap \partial p : \left(F_f = \emptyset \vee \exists e \in \partial F_v \cap \partial^{-1} v : |\partial^{-1} e \cap F_v| \neq 2\right)$, i.e., polyhedra are bounded in a vertex-manifold fashion.

(i) $\forall p \in P : \forall F_p \subsetneq \partial p : F_p = \emptyset \vee \exists e \in \partial F_p : |\partial^{-1} e \cap F_p| \neq 2$, i.e., polyhedra are bounded by exactly one closed shell.

($\mathcal{P}_{\text{IV}}$) Any two distinct n-cells are either not adjacent or adjacent via exactly one full third cell with lower dimension:

$$\forall n \forall c_1 \neq c_2 \in C_n : (\langle c_1 \rangle \cap \langle c_2 \rangle = \emptyset \vee \exists c_3 \in C_{n-1} : \langle c_1 \rangle \cap \langle c_2 \rangle = \langle c_3 \rangle)$$

($\mathcal{P}_{\text{V}}$) The mesh is a pure volume mesh:

(a) $\forall f \in F : 2 \geq \left|\partial^{-1} f\right| \geq 1$, i.e., each facet bounds one or two distinct polyhedra.

(b) $\forall e \in E : \left|\partial^{-1} e\right| \geq 1$, i.e., each edge bounds at least one facet.

(c) $\forall v \in V : \left|\partial^{-1} v\right| \geq 1$, i.e., each vertex bounds at least one edge.

($\mathcal{P}_{\text{VI}}$) The mesh is connected:

$$\forall C, C' \subsetneq \mathcal{P} : \mathcal{P} = C \cup C' \Rightarrow \langle C \rangle \cap \langle C' \rangle \neq \emptyset.$$

For our purposes the connectivity of concrete polyhedral meshes is defined via abstract polyhedral meshes; the specification of their geometry remains open for subsequent definition. The topology is the standard weak topology induced by interpreting $\mathcal{P}$ as a CW-complex [Hatcher 2002], with gluing maps determined by ∂ (up to homeomorphism). In an alternative view, it is the quotient topology induced by interpreting n-cells as closed n-balls (rather than open ones) and identifying all points on bounding elements shared between cells. Abstract tetrahedral and hexahedral meshes are realized as specializations of polyhedral meshes as defined below. Note that the below definition of abstract tet meshes is equivalent to that of a pure (abstract) 3D simplicial complex, a formalism commonly used in literature to describe tet meshes.

Definition 2.7 (Abstract tetrahedral mesh). We call $\mathcal{M} = P \cup F \cup E \cup V$ equipped with incidences ∂ an abstract tetrahedral mesh if it is an abstract polyhedral mesh and additionally:

(a) $\forall f \in F : |\partial f| = \left|\partial^2 f\right| = 3$, i.e., facets are triangles (2-simplices) bounded by a loop of three edges connecting three vertices.

(b) $\forall p \in P : |\partial h| = 4 \wedge \left|\partial^2 h\right| = 6 \wedge \left|\partial^3 h\right| = 4$, i.e., the polyhedra are tetrahedra (3-simplices), bounded by 4 distinct triangles, 6 distinct edges and 4 distinct vertices.

Definition 2.8 (Abstract hexahedral mesh). We call $\mathcal{H} = P \cup F \cup E \cup V$ equipped with incidences ∂ an abstract hexahedral mesh if it is an abstract polyhedral mesh and additionally:

(a) $\forall f \in F : |\partial f| = \left|\partial^2 f\right| = 4$, i.e., facets are quadrilaterals bounded by a loop of four edges connecting four vertices.

(b) $\forall h \in H : |\partial h| = 6 \wedge \left|\partial^2 h\right| = 12 \wedge \left|\partial^3 h\right| = 8$, i.e., the polyhedra are topological cubes, bounded by 6 distinct quadrilaterals, 12 distinct edges and 8 distinct vertices.

Definition 2.9 (Restriction). Let $C \subset \mathcal{P}$ be a set of cells of an abstract polyhedral mesh $\mathcal{P}$. A restriction of $\mathcal{P}$ to C is the abstract submesh $\mathcal{P}|_C := \langle C \rangle$ with incidences $\partial|_C$ induced by this restriction.

Note that such a restriction is not necessarily *pure* and not necessarily *connected*, hence it may itself not represent a polyhedral mesh as per our definition.

In a slight abuse of notation, we employ the same symbol ∂ that denotes cell-wise top-down incidences (the "bounds" of a cell) also to denote the boundary of an entire mesh, with a different meaning as defined below. From context it is always obvious whether ∂ is applied to cells of a cell complex or the cell complex as a whole, making the notation unambiguous in practice.

Definition 2.10 (Boundary). For any abstract polyhedral mesh $\mathcal{P}$, let F_1 be the set of all facets that are incident to exactly one polyhedron. We define the abstract *boundary* of the mesh $\partial\mathcal{P}$ as the abstract *polygonal* mesh $\mathcal{P}|_{F_1}$.

The polyhedral meshes investigated in this thesis are limited to compact, connected 3-manifolds with boundary that are embeddable in $\mathbb{R}^3$ without overlaps. Hence their boundary is a closed 2-manifold surface, i.e., each boundary edge is incident to exactly two boundary facets, and each boundary vertex has a disk-like star within the boundary.

Definition 2.11 (Dual). The *dual* of an abstract polyhedral mesh $\mathcal{P}$, is exactly the abstract cell complex $\mathcal{D}_\mathcal{P}$ obtained by

- identifying n-cells of $\mathcal{D}_\mathcal{P}$ with $(3-n)$-cells of $\mathcal{P}$,
- identifying ∂ of $\mathcal{D}_\mathcal{P}$ with ∂^{-1} of $\mathcal{P}$.

Note that $\mathcal{D}_\mathcal{P}$ has the properties of an abstract polyhedral mesh everywhere, except around the cells that are dual to the boundary cells of $\mathcal{P}$. Boundary facets, e.g., become edges in the dual that are bounded only from one side by a dual vertex. For simplicity, we consider elements dual to the boundary of $\mathcal{P}$ as *truncated* within $\mathcal{D}_\mathcal{P}$, rather than employing a cumbersome construction involving virtual ghost cells beyond the boundary. The incidence graph of $\mathcal{D}_\mathcal{P}$ is exactly that of $\mathcal{P}$ with all directed edges reversed and hierarchy levels turned upside-down. In later chapters we refer to *graph searches* on a mesh's dual; this refers to interpreting the node-vertex structure of the dual as a graph.

Non-trivial incidences Connectivity as defined above precludes multi-adjacency of cells via ($\mathcal{P}_{\text{IV}}$), e.g., two polyhedra sharing more than one face, as well as self-adjacency via ($\mathcal{P}_{\text{III}}$), e.g., a polyhedron adjacent to itself via one of its faces, as neither case can be represented with non-degenerate linear elements in $\mathbb{R}^3$. In some of the later chapters, these incidence conditions are explicitly relaxed to allow for more general cell complexes, along with handling element geometry in a way that avoids geometric degeneration.

Incidence ordering W.l.o.g. incidences are assumed to be canonically ordered whenever such an order exists:

- The cycle of edges and vertices bounding a facet,
- the cycle (sequence) of facets and cells around an interior (boundary) edge,
- the cycle of boundary edges and boundary facets around a boundary vertex, and
- in case of special polyhedra, the four (six) facets, six (twelve) edges and four (eight) vertices of a tetrahedron (hexahedron).

Furthermore two inverse incidence orderings for edges and facets, implying opposite directions, are associated to virtual *half-edges* and *half-facets*.

2.2 Tetrahedral Meshes

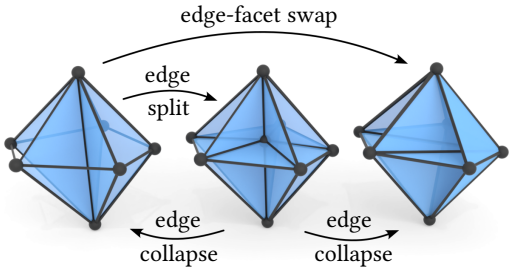
While generation of *hex* meshes is the goal of this thesis, the proposed pipeline (c.f. section 1.5.2) involves tetrahedral meshes as inputs and as the intermediate domain many of the algorithms in later chapters work on—directly or indirectly. Hence, a treatment of their structure, properties and construction is warranted here.

2.2.1 Connectivity

We consider the connectivity of tetrahedral mesh elements to be given as an abstract tet mesh $\mathcal{M}$ according to theorem 2.7. For notational clarity, we refer to the polyhedra of a **t**et mesh using the symbols $C_3 = T = \{t_1, t_2, \dots\}$. Unless otherwise specified, the geometric embedding is realized with linear elements in $\mathbb{R}^3$. This means only vertices are given explicit coordinates, edges are line segments, facets planar surface patches, and tetrahedra volumes, obtained via linear interpolation of vertex positions. The boundary of a tet mesh is a 2-manifold triangle mesh. Commonly, the task of tet mesh generation starts from such a triangle mesh as input, requiring its interior to be filled by a tet mesh.

An important aspect for algorithmic processing of tet meshes, also made use of in later chapters, is to consider mesh connectivity as dynamic and allow modifications where required, as explained below.

2.2.2 Modification Operators



The geometry of linear tet meshes is easily adjusted via vertex shifts—potentially accompanied by refinement to achieve higher resolution. Additionally, algorithms working on tetrahedral meshes often require updates to its connectivity. Due to the unstructured nature of tetrahedral meshes, such updates can be performed locally, without incurring any global connectivity updates. The building blocks of such local updates, each conserving a pure tetrahedral mesh, are the operators listed below.

Edge collapse The collapse of an edge is geometrically associated to the contraction of its two endpoint vertices onto one another. This degenerates the edge and, assuming linear elements, any facets and tets incident to it. Combinatorially, the connectivity can be updated accordingly, removing any elements that would be degenerated, and identifying with each other the remaining cells that collapse "onto" each other. Concretely, the two vertices of the collapsing edge, the two remaining edges of degenerating triangles incident to the collapsed edge, and the two remaining facets of degenerating tetrahedra are identified with each other, respectively. For a collapse to be admissible, local connectivity must satisfy certain, easily checked constraints [Dey et al. 1999].

Splits Splitting can be applied to edges, facets or cells, and means that an additional vertex is inserted, such that the element and any elements bounded by it are partitioned by the new vertex. In an edge split, the vertex is inserted along the edge, splitting it in two, and consequently also any facet and tetrahedron bounded by it. In a face(t) split, the vertex is inserted within the facet, splitting it and any incident tetrahedron in three. In a tet split, the vertex is inserted within the tetrahedron, simply splitting it into four parts. Additional elements are inserted in each of

these cases, guaranteeing that subelements obtained from splitting remain connected via shared bounds.

Flips A flip operation, akin to edge flips in triangle meshes, is not well defined for tetrahedral meshes. A close analogue are edge-face(t) swaps [Freitag and Ollivier-Gooch 1997]. "Swapping" an edge e for facets can be achieved by first splitting e and then collapsing any edge connecting the newly inserted vertex to a vertex from $\langle e^* \rangle \setminus \langle e \rangle^*$, i.e., from the previous cycle of vertices around e. This results in a triangulation of that cycle, the connectivity of which could be further adjusted by alternating edge splits and collapses. The inverse process of swapping such a triangulated cycle with an edge piercing through it can likewise be achieved as a sequence of splits and collapses.

2.2.3 Quality

Another important aspect for the processing of tet meshes is their geometric quality. In later chapters this is relevant especially when maps defined on tetrahedral meshes must be computed: The ease of finding good maps with numerical methods depends on the numerical condition of the mapping problem formulation, which often involves an abstract notion of input mesh quality, to be concretized in the following.

In terms of geometric quality, an ideal reference tetrahedron is the regular tetrahedron, i.e., one with all edges of equal length, 60° angles between edges and $\approx 70°$ dihedral angles between facets [Sorgente et al. 2023]. Because regular tetrahedra do not tile 3D space [Senechal 1981], let alone specific domains, mesh generation techniques can not achieve such ideal elements everywhere, which raises the question of how to measure and minimize deviations from this ideal. An often-cited source for different mesh quality metrics, commonly used as an inofficial standard reference, is the documentation of the Verdict library [Stimpson et al. 2007]. Shewchuk [2002b] gives a detailed overview of the mathematical connections between such metrics and the simulation-relevant aspects of interpolation errors and stiffness matrix conditioning, coming to the conclusion that each of these aspects are affected to different degrees by different metrics. Hence, there is no obvious choice for a single universal quality metric. A recent survey by Sorgente et al. [2023] gives an overview over quality metrics used in recent meshing literature, coming to a similar conclusion.

Practical meshing approaches are typically content with simply preventing (near-)degeneration of elements, and optimizing for—or at least reporting—any one measure that incorporates this notion [Shewchuk 2002b; Alexa 2019]. A reasonable choice for such a measure is the minimum dihedral angle per tetrahedron [Bern et al. 1995], also commonly used in recent literature.

Delaunay criterion It is a well-known fact that among all triangulations of a point set in $\mathbb{R}^2$, one that optimizes many relevant quality measures is one that satisfies the Delaunay criterion [Musin 1997]: The interior of the circumcircle of any triangle, i.e., the unique circle passing through each of the triangle's vertices, must not contain any other point of the input point set. While the Delaunay criterion can be extended easily to tet meshes in $\mathbb{R}^3$ by demanding that the circumsphere of any tet must be empty, only some of the quality guarantees from 2D carry over to Delaunay tetrahedrizations [Musin 1997]. Specifically, they may still contain so-called sliver tetrahedra [Cheng et al. 2000], that have very small or very large dihedral angles. For this reason (and others explicated below) there are many works concerned with post-processing of Delaunay tetrahedrizations after their initial generation.

2.2.4 Generation

Tet meshing, or tetrahedrization, is a topic extensively studied over the last decades. As in the scope of this thesis tetrahedral meshes are used merely as a precursor to the final hexahedral meshes, this section is meant to give only a minimal overview over the current state of the art. Specifically, it should convey that tet meshing up to high robustness and quality standards is established both theoretically and in practice, setting the (long-term) bar for hex meshing methods aspiring to be competitive in those regards and also justifying the choice of basing hex meshing pipelines on tet meshes as input or intermediate representations. For a more detailed and general literature overview, the reader is referred to recent works in the field [Cheng et al. 2013; Alexa 2020; Hu et al. 2020; Diazzi et al. 2023].

Delaunay tetrahedrization The vast majority of approaches to tet meshing are concerned with the Delaunay condition. If the input is just a point set in $\mathbb{R}^3$ the Delaunay tetrahedrization can be computed via incrementally inserting points of the input set into the current solution [Bowyer 1981; Watson 1981] and updating the tetrahedrization to maintain the Delaunay criterion. Such a tetrahedrization contains exactly the input vertices and no additional ones. However, when the input is a piecewise linear surface—a triangle mesh whose interior is to be meshed for example—a simple Delaunay tetrahedrization of its vertices generally can not preserve all input edges and facets, and thus often is a bad approximation of input geometry.

Constrained Delaunay Tetrahedralization On a 2D polygon one can compute a *constrained* Delaunay triangulation [Lee and Lin 1986; Chew 1987] that satisfies the Delaunay condition as much as possible while preserving input edges, specifically without introducing additional vertices. In 3D, not all piecewise linear surfaces even admit a tetrahedrization that conforms to input facets and edges without adding vertices that are not part of the input set [Schönhardt 1928], so called *Steiner* vertices. Even just determining how many Steiner points are sufficient for tetrahedrizing a given boundary mesh is NP-hard [Ruppert and Seidel 1992]. When arbitrary refinement of the original input is allowed for, the existence of a truly Delaunay tetrahedrization is guaranteed [Murphy et al. 2001; Cohen-Steiner et al. 2002], but regularly requires an impractically large number of additional vertices [Alexa 2020; Diazzi et al. 2023]. The most practical approach for generating tetrahedrizations that exactly conform to the input is that of *constrained* Delaunay tetrahedrization, which in general allows for insertion of additional vertices, but locally sacrifices the Delaunay criterion (near the boundary) to keep the number of such vertices manageable [Shewchuk 2002a; Si and Gärtner 2005; Diazzi et al. 2023]. Also related are remeshing and optimization approaches, that start from an initial tetrahedral mesh to then split, reconnect and/or shift elements in the vicinity of badly shaped tetrahedra, typically based on the Delaunay condition [Si 2008; Shewchuk and Si 2014] or measures closely related to it [Chen and Xu 2004; Alliez et al. 2005; Tournois et al. 2009; Chen et al. 2014; Feng et al. 2018; Alexa 2019].

Alternative techniques There also exist approaches not based on the Delaunay criterion. Advancing front methods initialize a "front" at the boundary mesh and then propagate it inwards [Cuillière et al. 2012; Alauzet and Marcum 2014; Haimes 2015], while grid-based methods typically cut the input with a uniform or adaptive grid, yielding easy-to-mesh interior grid cells and requiring special attention at the boundary [Labelle and Shewchuk 2007; Bronson et al. 2013; Doran et al. 2013]. Some other methods are concerned above all with tolerance for faulty input, e.g., self-intersecting surfaces, relaxing the requirement of the output tet mesh exactly conforming

to the input to be able to mesh such input at all [Shen et al. 2004; Mandad et al. 2015; Hu et al. 2020].

Robust software A plethora of robust, open-source tet meshing software packages exist today, implementing many of the above techniques. Among the most commonly used ones for academic research are TETGEN [Si 2015], CGAL's Mesh Generation package [Jamin et al. 2015], GMSH [Geuzaine and Remacle 2009], and NETGEN [Schöberl 1997]. Some codes accompanying recent research papers, such as TETWILD [Hu et al. 2018] and CDT [Diazzi et al. 2023], approach full practical robustness, reporting (near) 100% success rates on the challenging THINGI10K dataset [Zhou and Jacobson 2016].

2.3 Hexahedral Meshes

The goal of this thesis, simply speaking, is to generate "good" hex meshes. The very first questions to settle before going into the details of how to generate such meshes, is by which measures to define their structure and geometry, and based off that, what properties in those regards even make a hexahedral mesh "good." The above being established allows formulation of the desiderata of the core hex meshing problem in a precise manner, and finally to discuss existing approaches to solve it, as well as their trade-offs in terms of these desiderata. For an excellent, broader overview of hex-mesh-related aspects the reader is referred to the recent survey by Pietroni et al. [2022].

2.3.1 Connectivity

The connectivity of hex mesh elements is given as an abstract hex mesh $\mathcal{H}$ according to theorem 2.7. For notational clarity, we refer to the polyhedra of a **h**exahedral mesh as $C_3 = H = \{h_1, h_2, \dots\}$. The boundary of a hex mesh is a 2-manifold quad mesh. However, unlike triangulated boundaries for tet meshes, not every quad mesh can be the boundary of a hex mesh [Mitchell 1996]. A known necessary condition is that the number of boundary quadrilaterals has to be even, but this condition is only sufficient for hex mesh existence in the case of topological balls.

2.3.2 Geometry

The geometry per element of the hex mesh can be assessed via maps $\mathcal{I}$ from ideal reference elements onto the concrete per-element geometric embedding in $\mathbb{R}^3$. Each vertex v of the mesh must be mapped in a discrete way onto a specific point $\mathcal{I}_v \in \mathbb{R}^3$. Edges e are associated with a one-dimensional map from an interval onto the edges's concrete embedding: $\mathcal{I}_e : [0,1] \to \mathbb{R}^3$. Similarly, quadrilateral facets f and hexahedra h are embedded into $\mathbb{R}^3$ via maps from the ideal square $\mathcal{I}_f : [0,1]^2 \to \mathbb{R}^3$ and the ideal cube $\mathcal{I}_h : [0,1]^3 \to \mathbb{R}^3$, respectively. Unless otherwise specified, we assume the embedding map $\mathcal{I}$ is realized with (multi-)linear elements in $\mathbb{R}^3$. This means only vertices are given explicit coordinates, edges are line segments, facets ruled surface patches, and hexahedra volumes, recovered as linear, bilinear and trilinear interpolations of vertex positions, respectively. For a hexahedral element h with 8 corner vertices v_{ijk} for $i, j, k \in \{0, 1\}$ this gives rise to the so-called *geometric* map [Pietroni et al. 2022]:

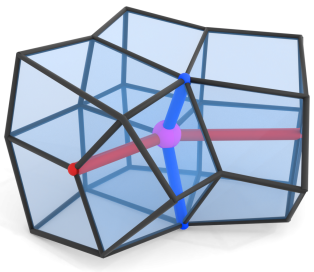
$$\mathcal{I}_h(u,v,w) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1} \mathcal{I}_{v_{ijk}} B_k(w) B_j(v) B_i(u), \qquad B_0(t) = 1 - t, \qquad B_1(t) = t. \tag{2.1}$$

More generally, one can construct $\mathcal{I}_h$ also as tensor products of higher-order one-dimensional basis functions B_i^n along each parametric direction i of the element, giving rise to (curved) higher-order

hexahedral elements. Due to this general property hexahedra are often referred to as *tensor-product* elements. The geometric map is furthermore often referred to as the *isoparametric* map, meaning the same interpolation functions of vertex (or nodal) attributes are used for geometry and any physical field variables defined on the element.

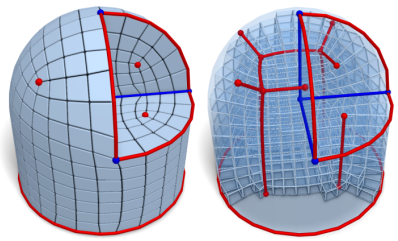
2.3.3 Structural Quality

Definition 2.12 (Singularities). The *valence* of a hex mesh edge is the number of incident hexahedra. An interior (boundary) edge is called *regular* if it has a valence of four (two), and *irregular* or synonymously *singular* otherwise. A vertex is called a singular node if it is incident to exactly one, three or more singular edges, *or* to two singular edges of different valence. Maximum chains of connected singular edges uninterrupted by singular nodes are called *singular links*, characterized by uniform valence.



While just the valence is sufficient to classify singular links, specifying types of singular nodes requires more complex descriptors. Nieser et al. [2011] recognized that (singular) node types in a hex mesh combinatorially correspond 1:1 to triangulations of the 2-sphere, while Zhang et al. [2023] demonstrated that via local modification operators any singular node can be decomposed into non-branching singular links, reducing the complexity of 3D nodes to a convolution of 2D links.

The entire graph defined by connectivity, valence and mesh embedding of singular links and nodes, is called the *singularity graph* or *singular graph*, is a key descriptor of the structural quality of a hexahedral mesh, as it captures differences in connectivity from an infinite cubic grid in which all edges are regular. The inset demonstrates this graph for a given hex mesh; singularities with above-regular valence are colored in blue and those with below-regular valence in red.



Opposites The notion of opposite elements is given in the following situations (where opposites are unambiguously determined):

- for pairs of quads bounding a hex,
- for pairs of edges bounding a quad,
- for pairs of quads incident on the same regular edge,
- for pairs of edges incident on the same regular node,
- for the pair of singular edges incident on a non-singular-node vertex.

Notably, there exists no general notion of opposites across singular edges or nodes.

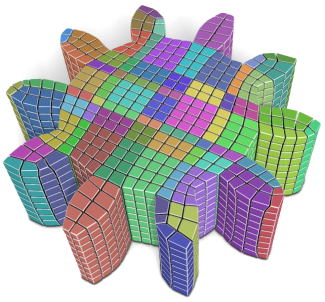
2.3.3.1 Block Structure

Definition 2.13 (Block decomposition). A block decomposition of a hexahedral mesh $\mathcal{H}$ is a set $B = \{b_1, b_2, \ldots, b_{|B|}\}$ of blocks $b_i \subset H \subset \mathcal{H}$ consisting of the mesh's hexahedra, such that

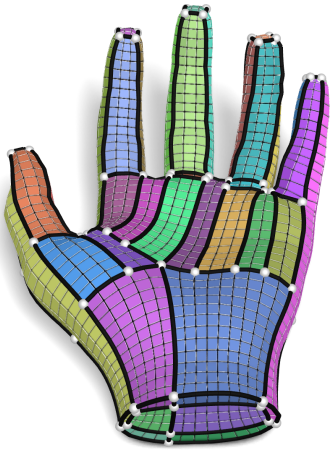
1. Blocks are disjoint: $\forall b_i \neq b_j \in B : b_i \cap b_j = \emptyset$.
2. B covers all hexahedra: $\bigcup_{b \in B} b = H$.

3. Individual blocks are equivalent to cuboid cutouts of the regular grid, i.e., the restriction of $\mathcal{H}|_b$ to any block b has only regular interior edges and the singular graph induced by this restriction has 12 edges and 8 vertices that partition the block boundary into 6 quadrilaterals.

We observe that any hexahedral mesh admits such a decomposition, giving the trivial one in which each hex is an individual block as an example. Any such decomposition can be viewed as a disjoint, complete grouping of hexahedra into larger hexahedra. This grouping is illustrated via coloring in the inset. If these larger hexahedra, with vertices, edges and facets induced by the singular graph within their restriction, together again form a combinatorial hexahedral mesh in the sense as defined above, i.e., any two blocks share nothing or exactly one vertex, full edge or full facet, then we call the decomposition *conforming* (as in the inset), otherwise it is *non-conforming*.



T-mesh In fact, even the non-conforming decomposition can be modeled as a mesh again, but only with the general connectivity of a polyhedral combinatorial complex, defining any overlap between the sides of two blocks as an individual facet, and any overlap between the sides of two facets as an individual edge, as illustrated in the inset. While the connectivity of a polyhedral mesh alone can not capture that each polyhedron still implicitly is a cuboid hexahedron (just with more facets, edges and vertices), we can enrich the combinatorial structure by the additional information about where facets or edges meet in a "T"-shape, or in other words, marking face and dihedral angles as either 90° or 180°. A combinatorial polyhedral mesh that is enriched by such markers, and describes a non-conforming cubic block decomposition, is what we henceforth call a *T-mesh*. T-meshes are a crucial tool in later chapters, as they can be constructed not only from hex meshes but also from parametrized volumes, i.e., the precursors to hex meshes in our pipeline (more on this in chapters 5 and 6).



Base complex The *base complex* of a hexahedral mesh is its (unique) coarsest conforming block decomposition. There is a straightforward construction routine for the base complex of a given hexahedral mesh $\mathcal{H}$: For each singular edge, mark all incident facets as block walls and propagate this marking, always marking the facet that is opposite the last one across a shared regular edge, until reaching another singular edge or the boundary [Brückler et al. 2022b]. Any maximum set of hexahedra connected via non-wall facets then forms a block of the base complex.

2.3.3.2 Mesh Regularity

Consider the following extremes: A perfectly structured (or *regular*) mesh that is combinatorially equivalent to a cuboid cutout from the regular grid, and an unstructured (or *irregular*) mesh which in turn could be obtained from a tetrahedral mesh by splitting each tetrahedron into four hexahedra in a conforming way. The difference between the two can be characterized based on both the singular graph and the base complex of these meshes. In the irregular case the singularity graph is very dense with respect to the overall mesh resolution. Consequently, the base complex is

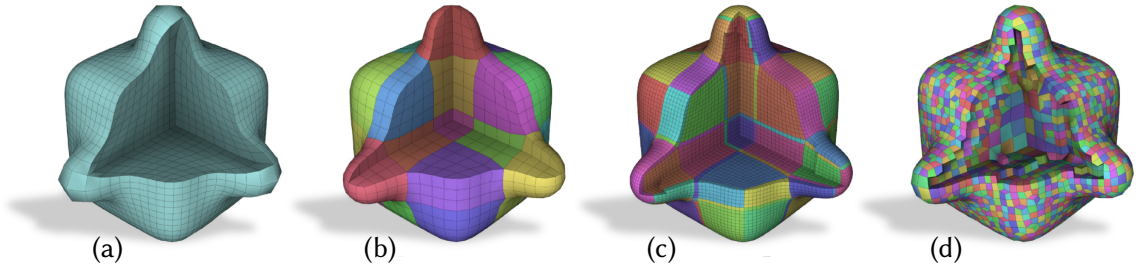


Figure 2.2: Different degrees of hex mesh regularity, as indicated by coloring of blocks of the base complex. (a) Regular, the connectivity is equivalent to that of a regular grid. (b) Semi-regular, with a simple singularity structure and a minimal base complex for this structure. (c) Semi-regular, with a simple singularity structure, but misaligned singularities and thus a non-minimal base complex. (d) Irregular, with very dense singular graph. (Image adapted from Pietroni et al. [2022, Fig. 1])

very fine, with many small blocks, i.e., a relatively high ratio of base complex blocks to hexahedra in the mesh. In the regular case, both the singularity graph and the base complex are as coarse as can be, with only the bare minimum of singular edges on the boundary, and a single block in the base complex. Overall, it is still only seldom a good choice, as the elements of a perfectly regular hex mesh must undergo large distortions if the mesh is to match a non-trivial shape, and distorted elements conversely have a detrimental effect for applications (cf. fig. 2.2(a)).

Therefore, cases between the two extremes considered above are most relevant in practice, pushing structural simplicity only as far as element geometry does not suffer. To describe such cases, for which the singular structure is relatively simple and where the mesh contains only few irregular elements compared to the overall element count, the term *semi-regular* has been established [Pietroni et al. 2022]. To be even more precise, the term *valence semi-regular* is meant for meshes with a simple singular structure but unnecessarily many blocks in the base complex, due to singularities not being *aligned* on shared sheets of quadrilaterals within the mesh. A term commonly used for semi-regular meshes that additionally also have a simple base complex is *block-structured*. A standard finite-element method, which works only on a per-element level, might not benefit from such semi-regular structuring, but several related applications like multi-grid methods [Wesseling and Oosterlee 2001], isogeometric analysis [Hughes et al. 2005] and mesh compression [Isenburg and Alliez 2002; Tautges 2004] can use it to their advantage (c.f. section 2.3.5). Furthermore a maximally coarse, conforming block decomposition may itself be the ideal domain for some techniques, as may be the case for *spectral element methods* [Kopriva 2006; Canuto et al. 2007; Kopriva 2009] of high polynomial degree [Weiss et al. 2023], which mix the globally supported nature of the spectral method with the locally supported, but geometrically flexible finite element method. For the meshing pipeline treated in this thesis, generating block-structured meshes is an explicit goal, in line with such requirements.

2.3.4 Geometric Quality

There are two central aspects of the geometry of a hex mesh that have a direct impact on applications: The *accuracy*, i.e., how closely an input shape and its features are recovered by a hex mesh generated from it, and the *element quality*, i.e., how well individual hexahedra match ideal element shapes.

2.3.4.1 Accuracy

It is obvious how accuracy can affect the accuracy of, e.g., simulations. The results of a simulation modeling a geometric ball with a single cube is of little value, as the distance of individual points

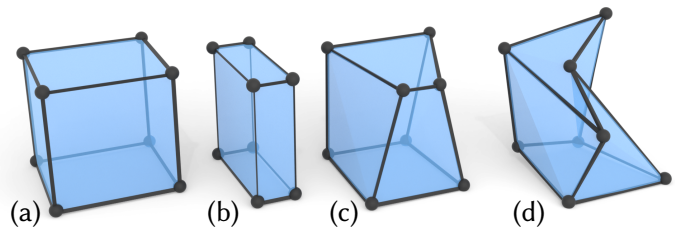


Figure 2.3: Hexahedra of varying quality: (a) an ideal cube, (b) An anisotropic cuboid, (c) a moderately deformed cube, (d) a deformed cube with partially sign-inverting geometric map $\mathcal{I}$. In most applications, (a) or (b) are desired, while (d) is inacceptable; tolerance for (c) is application-dependant.

on the respective boundaries are too high to allow for meaningful result transfer between the domains. Likewise, concerning the preservation of features, approximating a sharp feature crease in the input with staircase artifacts of a regular cubic grid hampers the usefulness of the mesh for correctly predicting, e.g., airflow around that crease. Despite these obvious detrimental effects of bad accuracy, it is an aspect that is rarely accounted for explicitly—neither through algorithmic guarantees nor through verification of the output via, e.g., the Hausdorff distance. Some mesh generation algorithms have the property that accuracy increases with target mesh resolution; this at least guarantees that arbitrary accuracy *can* be achieved, but depending on the method this might incur slowdowns due to high element counts. Therefore, methods that include the notion of accuracy by construction, e.g., via parametrizations aligned to both features and boundaries, are desirable.

2.3.4.2 Element Quality

Like for tetrahedral meshes, it is quite well understood that the quality of individual elements is linked to the performance and accuracy of numerical simulation methods performed on them [Zlámal 1968; Ciarlet 2002; Shewchuk 2002b]; however, the concrete relation and quality requirements depends on both the treated problem and the employed numerical solver [Pietroni et al. 2022]. A measure that appears almost universally relevant is the notion of element *validity*, typically expressed via the Jacobian of the geometric map (eq. (2.1)) between a reference cube and the geometric hex element. Concretely, finite-element methods require the determinant of the Jacobian matrix to be positive everywhere [Mitchell et al. 1971; Salagame and Belegundu 1994; Barrett 1996; Knupp 2000], which intuitively corresponds to the notion of volume portions of the reference cube not sign-inverting or folding over each other when deforming it into the actual hex. The Jacobian determinant, if not simply used as a threshold, is a general measure of volume distortion between the reference and the actual element, but does not capture angle distortion. Other requirements reported for different application scenarios are avoidance of large angles [Zlámal 1968; Ciarlet 2002; Shewchuk 2002b], facet planarity [Beirão da Veiga et al. 2014] and mutual orthogonality of primal facets and dual edges [Moraes et al. 2013; Aqilah et al. 2018].

Jacobian of hex elements Quality concerns in the scope of this thesis mostly revolve around the Jacobian determinant of the geometric map (c.f. eq. (2.1)), $\det J_{\mathcal{I}}$. This is due to both the universal relevance of element validity for applications and the fact that the Jacobian of the geometric map and that of precursor parametric maps in our approach are closely related (though not equivalent for linear elements), allowing control over element quality at the parametrization level. While the Jacobian determinant $\det J_{\mathcal{I}}$ can easily be evaluated at any point within the hex, its exact minimum $\min_{[0,1]^3} \det J_{\mathcal{I}}$, determining element validity, generally has no closed-form expression. Beside iterative methods that refine bounds for $\det J_{\mathcal{I}}$, for trilinear elements the convex problem of finding $\min_{[0,1]^3} \det J_{\mathcal{I}}$ has been shown to be equivalently replaceable by a convex problem [Marschner et al. 2020] with coinciding minimum. While sampling the

determinant at certain points in the volume is common practice, as demonstrated by works commonly reporting the Jacobian or Scaled Jacobian metric from the Verdict library [Stimpson et al. 2007], it is prone to overestimations of the true minimum [Knupp 1990] and thus risks elements being falsely reported as valid.

2.3.5 Benefits over Tetrahedral Meshes

One primary motivation for using hex over tet meshes is the potential for higher accuracy and efficiency in numerical simulations. This advantage is not necessarily a general property of hex meshes, but rather depends on the specific mesh, application and simulation method. In general, it is not hard to find publications that simply observe a (sometimes only slightly) better accuracy or convergence rate of simulations running on hex rather than on tet meshes (see, e.g., Benzley et al. [1995], Bourdin et al. [2007], Tadepalli et al. [2011], and Thieulot and Bangerth [2025]). Below are more concrete, evidence backed benefits of hex meshes over tet meshes.

Element count In structured regions, hex meshes need about a quarter of the volume elements to mesh a set of vertices as compared to tetrahedral meshes [Tuchinsky and Clark 1997]. Of course, this saving in element count can only be translated into computational savings if the time spent for generating such a structured hex mesh is comparable to that for generating a tet mesh; alas, this is currently not the case [Tautges 2001; Sarrate Ramos et al. 2014; Pietroni et al. 2022]. However, the upfront cost of hex meshing may be amortized in the long run, if the same mesh is used in multiple simulations or very time-consuming ones. Note also that tet meshes can more easily adapt resolution to local features or sizing constraints, potentially reducing element counts when uniform resolution is not required.

Linear vs. trilinear There is a subtle difference between what is commonly referred to as "linear" elements in tet and hex meshes. Hexahedra are *trilinear* elements offering a space of trilinear basis functions for interpolating physical field variables, while tetrahedra offer only a linear basis per tet. It is a well-known fact that the lower amount of degrees of freedom of linear tets introduces artificial stiffness in finite-element simulations, a phenomenon known as *locking* [Wang et al. 2004], whereas neither higher-order tets nor (tri)linear hex elements suffer from this effect. Schneider et al. [2022] therefore advocate for the use of second-order tet meshes as a general-purpose simulation domain. However, in certain simulation scenarios where time integration is performed explicitly, going to higher-order bases is not feasible as it would require a (potentially drastic) reduction in the time step to achieve numerical stability [Courant et al. 1967; Weber et al. 2022]. This for example applies to crash simulation [Gravouil et al. 2009] and surgical simulation [Gao and Peters 2021], as well as related simulations of fast transient dynamic processes or hyper-elastic and plastic processes, respectively [Pietroni et al. 2022].

Tensor product structure Hexahedral meshes exhibit a two-fold tensor-product structure: First, the geometric map of each hex element is a trivariate tensor product of univariate basis functions (eq. (2.1)). Second, in block-structured meshes, blocks themselves can be viewed as tensor products of one-dimensional element chains along each parametric direction of the block. This general tensor-product nature corresponds well to that of spline elements used in, e.g., isogeometric analysis [Hughes et al. 2005], allowing hex meshing techniques to be applied in spline modelling [Cottrell et al. 2006], as demonstrated in several works [Wang et al. 2012; Yu and Wei 2020].

2.3.6 Hex Meshing Desiderata

Before discussing and assessing existing approaches to hex meshing, it is important to first clearly formulate the desiderata for these methods. This allows to then systematically compare different methods in terms of how well they fulfill these desiderata, and to identify open challenges in the field.

Input specification The method should handle as input volumetric domains in R^3

(Arbitrary topology) of arbitrary topology,

(Arbitrary geometry) of arbitrary shape,

(Target resolution) with specification of desired mesh resolution (uniform or adaptive),

(Features) optionally with marked feature points, curves and/or surfaces.

Output specification The method should generate a hex mesh that

(Accuracy) matches the input's topology and faithfully approximates its boundaries and features,

(Quality) has only *valid* elements (positive det J_I everywhere), and maximizes application-dependent quality criteria,

(Structuredness) exhibits a semi-regular structure, induced by an adequately simple and well-aligned singularity graph,

(Sizing) matches the target resolution.

Process specification The method should

(Automation) require minimal user input and be intuitive to use for non-experts,

(Robustness) reliably produce the specified output without algorithmic or numerical failures,

(Efficiency) proceed fast enough to not act as a bottleneck in application pipelines.

The specifications of output and process are somewhat vague in this general formulation, but can be filled with more concrete meaning based on the specific application scenario and by comparison to the state of the art in tet meshing, which can serve as a benchmark in terms of accuracy, automation, robustness and efficiency.

2.3.7 Hex Meshing Simplification Paradigms

The general problem of hex meshing, as formulated above, is in essence a mixed-integer problem with strongly coupled integer, combinatorial and continuous degrees of freedom (c.f. section 1.4), a large set of complex constraints and multiple, sometimes conflicting objectives. Despite a long history of research in various disciplines, including engineering, computer graphics, and computational geometry, no all-in-one solution exists today that fulfills the above desiderata to a satisfactory degree—especially when measured against what is possible in tet meshing (c.f. section 2.2.4) [Pietroni et al. 2022]. Rather, the current state of the art encompasses a wide landscape of different hex meshing algorithms that each fulfill some of the desiderata, but none that fulfills all of them. Typically, the type of algorithmic approach rather than its concrete implementation already determines—explicitly or implicitly—a certain compromise in fulfilling the above requirements, as outlined in sections 1.4 and 1.5. Below, algorithmic classes and

concrete instances are discussed in terms of which desiderata they satisfy well, and which ones they struggle with. For a similar, more exhaustive comparison of approaches, the reader is referred to Pietroni et al. [2022, specifically table 2] .

2.3.7.1 Direct vs. Indirect

On a fundamental level, approaches can be divided into two classes: Those that *directly* apply the hex meshing algorithm to an input boundary specification, and those that employ an *indirect* route, first building an intermediate tet mesh of the domain and then *remeshing* it into a hex mesh. Direct approaches often assume a boundary quad mesh is given and then find a hex mesh that exactly conforms to it—as long as one exists, which is not the case for all quad meshes [Mitchell 1996]. One challenge of direct approaches therefore is to avoid intermediate local quad mesh configurations that cannot be filled with a hex mesh. In the following we refer to configurations as either *meshable* or *unmeshable*, implicitly referring to a hex mesh specifically. Even for meshable boundary quad meshes with very few elements, finding a hexahedralization can be surprisingly hard, e.g., for the instance *Schneiders' pyramid* [Schneiders 1996], a pyramid with four quads tiling its square base and three quads each on its triangular sides (cf. section 1.1). The success of direct methods often hinges on their ability to find well-structured, high-quality hex meshes for similarly challenging configurations, when they appear as local subproblems of the global problem.

The naive approach to *indirect* hex meshing of simply generating a tet mesh and then splitting each tet into four hexes [Li et al. 1995] inherits input handling, robustness, efficiency, and automation from tet meshing but generates unstructured output of little practical value. Hence, current indirect approaches tend to only use the tet mesh as a volumetric scaffold on which additional intermediate data is computed and stored. This incrementally enriched data can later be mapped to the degrees of freedom of the final hex mesh. The main challenges of indirect approaches are to translate between such intermediate data and a valid hex mesh, as well as filtering out unmeshable intermediate representations.

2.3.7.2 Advancing Fronts

A class of direct methods developed in the earlier years of hex meshing are *advancing front* methods. Early methods started from a boundary quad mesh and expanded this boundary front inwards by incrementally inserting hexahedra at the current front, until only internal voids matching predefined patterns remain, which are then filled via templates [Blacker and Meyers 1993; Blacker 1996]. The low robustness of such methods can be increased by employing a strategy to mediate between different parts of the front when they geometrically collide during expansion [Owen and Saigal 2000]. However, a more glaring issue is the inability to fully avoid cavities in the process that are either downright unmeshable [Mitchell 1996] or require low-quality elements. Even more recent methods that start from more general boundary representations for more flexibility [Staten et al. 2010] still suffer from significant robustness issues. The *receding-front* method [Ruiz-Gironés et al. 2012] is applicable to inverse domains, where a quad-meshed cavity (representing an object immersed in a medium like air or water) is given and expanded outwards towards an unmeshed outer boundary. This algorithm is robust but severely limited in terms of admissible inputs, allowing only immersed topological spheres without sharp features to be meshed. In general, advancing front methods are useful when exact conformity with a given quad mesh is required, e.g., at interfaces of individual objects of a larger assembly, but offer limited robustness, structuredness and handling of varied inputs.

2.3.7.3 Dual Methods

Dual methods do not constitute a single, well-defined class of methods. Rather, the term refers to methods of potentially different algorithmic approaches, whose construction rules revolve around the dual of a hex mesh, rather than the primal. For example, dual advancing-front methods interpret the advancing-front process of incrementally adding hexahedra as incrementally fixing the intersections of three sheets in the hex mesh dual [Tautges et al. 1996]. This view allows a more robust, combinatorial treatment of the advancing-front mesh construction process [Folwell and Mitchell 1999; Ledoux and Weill 2008], but struggles especially with the handling of self-intersecting dual sheets. Beside robustness issues related to such sheet self-intersections, methods tend to often yield low-quality or even invalid elements. Other methods seek to gain advantage from the *semi-local* nature of the dual, meaning that the more global restrictedness and relation of dual sheets is used to preclude unmeshable configurations a priori [Müller-Hannemann 2004; Kremer et al. 2014]—rather than by backtracking from the deepest level. In practice this leads to improved robustness and element quality, compared to (dual) advancing-front methods. Again, dropping the restriction of exactly conforming to a boundary quad mesh widens the admissible solution space and allows more flexibility for methods to obtain better results. Takayama [2019] combine this relaxation with user guidance of dual sheets, but the method is expected to require much higher turnover times, due to both user interaction and a computationally expensive formulation. Livesu et al. [2020] use principal curvature directions of the boundary surface as guidance for boundary loop curves that bound implicit dual sheets, but can not guarantee that the generated mesh is a pure hex mesh.

2.3.7.4 Decomposition-Based Sweeping

The method of *sweeping* originates from works on CAD models predominantly formed as extrusions of 2D shapes. For such models, an intuitive aproach is to generate a quad mesh of the 2D shape and then sweep it from one end of the volume towards the other end, creating well-spaced layers of hexes with identical connectivity in between [Shin 1996]. This works well for models, where two opposite *source* and *target* patches on the domain boundary can easily be identified [Liu and Gadh 1997; Liu et al. 1999], but can not handle more complex input geometries. Handling more complex models requires more advanced schemes to sweep between multiple sources and targets, a decomposition of the domain into chunks for which sweeping is applicable, and a way to ensure that the generated meshes are compatible between chunks. The main challenge lies within the decomposition step. A plethora of decomposition schemes have been studied in literature, some based on unreliable heuristics [White et al. 2004; Wu and Gao 2014; Wu et al. 2018], some revolving around user interaction [Lu et al. 2017], some based on medial axis shape descriptors but restricted to certain input shapes [Price and Armstrong 1997; Sheffer et al. 1999; Zhang et al. 2007; Livesu et al. 2016, 2017]. A robust, automatic algorithm for the decomposition of general shapes into sweepable volumes remains elusive. Current commercial software packages for hex meshing, such as ANSYS [ANSYS Inc 2025] or HYPERMESH [Altair Inc 2025], base their workflow on manual block decomposition by an expert user, followed by automated, high-quality sweeping of the resulting blocks.

2.3.7.5 Grid/Octree-Based

A very straightforward way of hex meshing a given volume is overlaying it with a 3D voxel grid, including in the mesh all voxels in the volume's interior and projecting vertices on the mesh boundary onto the volume boundary [Schneiders 1996]. Accuracy in terms of both geometry and

topology requires that the grid is chosen fine enough to capture all topological and geometric features. To circumvent the associated cubic growth in grid size, methods switched to adaptive grids as a baseline. This, however, requires refinement schemes to produce a conforming hex mesh from the non-conforming adaptive grid. The other important ingredients are the grid construction, including a strategy to minimize the number of sites requiring refinement (called *hanging nodes*), and the boundary projection mechanism, including feature recovery. Adapting the grid so that it can preserve the input domain topology is still a hard problem, and many methods can not guarantee this [Maréchal 2009; Livesu et al. 2021; Pitzalis et al. 2021] or can only achieve it via excessive refinement [Gao et al. 2019].

In terms of grid construction and hanging node minimization, methods that operate on octrees and their duals have proven superior over those working with other adaptive grid types. This is due to the fact that refinement schemes provably handling all possible hanging node configurations have only been found for the former. An important observation, paving the path for dual grid methods, was made by Maréchal [2009], who recognized that constraining the valences of grid edges and grid vertices guarantees that the grid dual is a pure hex mesh. Later methods [Hu et al. 2013; Gao et al. 2019; Livesu et al. 2021] enforce these constraints in an over-restrictive manner, resulting in excessive refinement, as demonstrated by Pitzalis et al. [2021] who are able to reduce element counts by half.

The most delicate part of grid-based algorithms is the handling of boundary and features, the main challenge being to preserve valid hex elements throughout the boundary fitting process. To this end, some methods employ vertex-based smoothing [Lin et al. 2015; Livesu et al. 2021], which sometimes fails to correctly reproduce the input shape, while the method of Gao et al. [2019] employs a more complex deformation scheme that is able to guarantee both topological and geometric accuracy, as well as hex element validity, but comes with computational and memory costs potentially overtaxing most current hardware. The feature preservation routine of the latter method works well in practice, but lacks guarantees, especially in cases of regions where multiple feature curves coincide.

Overall, methods from this class are the most robust, with current ones able to automatically process arbitrarily complex input without failure. For this reason, they are the only automatic method implemented in current commercial hex meshing softwares [Coreform Inc 2025]. The great robustness, however, is paid for by accepting poor (independent) resolution control, severe unstructuredness and poor geometric quality at the resulting mesh boundary.

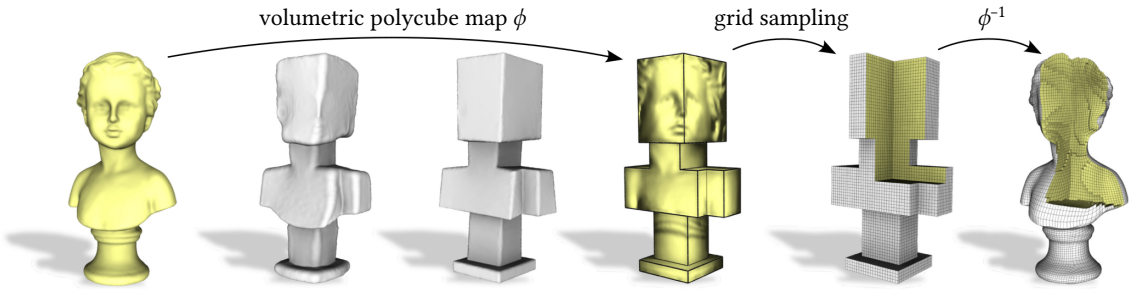


Figure 2.4: The core idea of polycube hex meshing: A map ϕ deforms the volume bounded by the input shape into a polycube. In the image, each boundary facet's normal is aligned with one of the three principal axes, so that boundaries lie on integer planes of a cartesian grid. This grid is sampled within the polycube, and the inverse map ϕ^{-1} pulls back the grid nodes into the input shape, forming a hex mesh of it. (Image from Pietroni et al. [2022])

2.3.7.6 Polycubes

Polycube approaches can be viewed as generalizing the idea of overlaying a grid with the input volume. However, rather than overlaying the grid with the raw input, which introduces the problem of conciliating the inner grid cells with the domain boundary in a post-process, polycube methods preemptively deform the input shape so that its boundary exactly aligns with the voxels of a uniform cartesian grid. Because the so deformed input has the shape of multiple cubes glued together, it is called a *polycube*. The deformation between the input shape and its polycube is represented as a map between the two, called *polycube map* [Tarini et al. 2004]. To be able to transport the grid overlayed on the polycube back into the input shape, so that it forms a hex mesh of the input, the polycube map needs to be volumetric, i.e., prescribe how points within the polycube (which lies in 3D *parameter space*) correspond to points within the input (which lies in 3D *object space*). This is achieved by first tetrahedrizing the input and computing a simplicial map into parameter space for each tetrahedron, implemented via assigning 3D parametric coordinates to each tet mesh vertex, and recovering the parametric shape of higher-dimensional elements via linear interpolation. Any grid node can then be pulled back along this simplicial map by determining in which parametric tet it is contained, computing the barycentric coordinates within this tet and recovering the object space position by barycentric interpolation of object space coordinates.

The polycube mapping problem can be interpreted as two-stage process: In the first stage, only the target polycube boundary shape is determined, in the second stage the volumetric simplicial map between input and the filled polycube is computed. These stages can but do not have to be clearly separated in practice; some methods determine the target shape only as part of the map optimization process. Still, this division is instructive for differentiating between the disparate challenges in determining a valid polycube boundary structure and computing a valid volumetric map.

Polycube segmentation The combinatorial structure of a polycube map can be expressed by a polycube *labeling* or *segmentation*, which means assigning to each part of the input boundary—for example per triangle of a triangle boundary mesh—a target normal axis from among $\{\pm X, \pm Y, \pm Z\}$. Under this encoding, locally connected triangles with the same label lie on the same facet of a cube-shaped component of the polycube. An intuitive first observation is that labelings can be invalid; for example, having the label $+X$ surrounded by only labels $+Y$ is not meshable, but much more complicated constraints arise [Eppstein and Mumford 2010; Livesu et al. 2013; Mandad et al. 2022]. Algorithms that correct unmeshable labelings based on local conditions [Gregson et al. 2011; Livesu et al. 2013] fall short of guaranteeing convergence to a valid labeling, making global conditions necessary, which are generally quite hard to establish [Sokolov and Ray 2015; Sokolov 2016; Mandad et al. 2022]. An interesting recent line of works is the characterization of polycube segmentations via dual loops traversing a polycube's quadrilateral facets [Baumeister and Kobbelt 2023; Snoep et al. 2025a], which has also spawned an algorithm that given a maximally coarse polycube segmentation can automatically transform this into a more refined labelling while maintaining validity [Snoep et al. 2025b]. An open problem remains how to automatically initialize a maximally coarse polycube for higher-genus surfaces, solved in the above approach via user interaction, as is commonly employed also in other polycube segmentation approaches [Yu and Wei 2020; Li et al. 2021].

Injective polycube maps There exist two types of approaches for the computation of polycube maps: Those that deform the tetrahedrization of the input incrementally until it matches a labelling

computed a priori [Gregson et al. 2011; Livesu et al. 2013] or on the fly [Huang et al. 2014a; Fang et al. 2016; Fu et al. 2016; Mandad et al. 2022], and those that compute a map between the input boundary and final polycube boundary first and try to find a volumetric map matching that boundary specification afterwards [Li et al. 2021]. The first type of approaches can more easily ensure map injectivity, but typically can only deal with structural defects appearing in the deformation process in a heuristic manner, while the second type of approaches shares the difficulties of general volumetric map generation, a topic discussed in further detail in section 3.6.2.

Polycube Quantization Beyond the assignment of direction labels for boundary patches, each such patch also requires sizing information in terms of the lengths of its two orthogonal sides. Assuming that the grid in parameter space is exactly sampled at integer coordinates, this means that any surface patch of a polycube must lie on an integer plane, i.e., the patches' constant (normal) coordinate must be integer. The process of assigning these integers is called *quantization*. A common approach to solve the mixed-integer problem of computing a quantized polycube map is to first compute the continuous relaxation and then fix the integer degrees of freedom so that they closely match their continuous optima. Historically, a common approach for this quantization step has been to simply round the continuous coordinates to their nearest integer [Gregson et al. 2011; Livesu et al. 2013], which is heuristic in nature and can force the map into degeneration, e.g., when two patches orthogonally adjacent to another patch are snapped to the same integer. Specific quantization strategies for polycubes have been devised as robust replacements for rounding [Cherchi et al. 2016; Chen et al. 2019; Protais et al. 2022]. The goal of such methods is the same as the core goal of the methods devised later in this thesis; however, due to targetting more general *integer-grid maps*, of which polycube maps are only a small subclass, our methods are inherently more complex. Our method can be applied also to polycube maps, while the inverse is not true for polycube quantization methods.

Limitations and generalizations Polycube map approaches to hex meshing—relative to other approaches—currently sit at a sweet spot between many of the aforementioned desiderata. Current methods allow automatic meshing of large datasets of moderately varied input models, are able to generate hex meshes with mostly semi-regular block structure and moderately high quality (via the use of well-studied map optimization techniques), while succeeding to produce valid and feature-aligned output most of the time. However, there are inherent limits to the expressivity of polycube maps: Not every hex mesh can be obtained via a polycube map. In fact, polycube maps can only produce hex meshes with singularity graphs of limited valence fully embedded in the mesh boundary (this is implied by the definition of the polycube target shape), and also with limits on handling higher-genus topologies (this is implied by only allowing deformation onto the target polycube but no topological cuts). Surpassing the latter limitation by allowing virtual cuts only to split closed handles, as done in some recent works [Fang et al. 2016; Mandad et al. 2022], is essentially a partial step towards the domain of more general *integer-grid maps*. The former limitation of a restricted singularity graph also poses a restriction on features that can be preserved by polycube maps; a concave boundary vertex of a polycube-induced hex mesh can have at most three edges, but what if there are more incident feature curves in the input? The restriction on singular structure is another one that is lifted by transitioning over to more general integer-grid maps, that allow interior singularities induced by constrained, partial cuts through the volume before mapping, and (non-local) self-overlaps in parameter space. In essence, compared to polycubes, integer-grid maps offer a similar map-based approach to hex meshing, sharing most of the beneficial properties—and some of the difficulties—related to this, but are truly unrestrictive: There exists an integer-grid map for every possible hex mesh. This

increased generality allows finding better-structured, higher-quality solutions with arbitrary feature alignment, in cases where such would be cut off by more restricted approaches. The obvious downside has been that the increased complexity of this general setting—so far—has stalled the development of truly automatic, robust and efficient solutions. An extensive study of integer-grid maps follows in the next chapter.

2.3.7.7 Post-Processing

A substantial number of recent publications is concerned with improvements of hex meshes as a post-process, i.e., after their generation rather than as part of it. Most relevant in achieving the above desiderata are approaches targeting structure simplification and element geometry optimization.

Structure simplification methods are mostly concerned with reducing the number of base complex components. Operations employed to achieve this typically involve the collapse of entire sheets of hexahedra, requiring additional refinement to preserve resolution and incurring a loss in accuracy (unless a representation of the original input domain is used for projection). Using such operators, methods preserving the singular structure and simply optimizing alignment [Gao et al. 2015a] as well as methods simplifying the singular structure [Gao et al. 2017b; Xu et al. 2021] have been suggested. The latter methods have been demonstrated to yield meshes of relatively simple structure, even for unstructured inputs obtained from octree-based hex meshing. However, simplification is performed in a greedy manner and uses a restricted set of operators, and thus is likely to end up in local minima, warranting a focus on structuredness already at the mesh generation stage.

The main goal of geometric optimization is to reduce the number of invalid or low-quality elements, as well as improving the global lower bound of the employed quality metric—typically the Jacobian determinant of the geometric map. All such geometric optimization methods are rather universal in nature, and front-to-end hex meshing pipelines typically employ at least one such method as a post-process. In the vast majority of cases, the mesh connectivity is considered fixed, and only vertices are shifted; this admits a continuous optimization formulation, where the quality metric is interpreted as an energy functional of vertex positions to be optimized. Due to this energy function being highly non-linear and often non-convex, a global minimum can not be reliably found. Available methods therefore are faced with devising proxy energy functions for which good minima are easier to find via optimization, as well as devising (heuristic) optimization strategies that avoid bad local minima. Optimization schemes employing global vertex position updates [Yilmaz and Kuzuoglu 2009; Gao and Chen 2016] generally seem to exhibit inferior performance compared to methods employing Gauss-Seidel iterations to shift vertices one at a time [Knupp 2003; Wilson et al. 2012; Ruiz-Gironés et al. 2014, 2015; Vartziotis and Papadrakakis 2017]. The latter methods can explicitly prevent a local worsening of quality or invalid elements, but are more likely to converge into local minima early in the process. Hybrid local-global schemes [Maréchal 2009; Livesu et al. 2015; Xu et al. 2018] are a promising, more recent avenue that so far has been shown to outperform the aforementioned approaches in terms of achieving higher quality, or, in other cases, a better balance between accuracy and quality improvements.

Integer-Grid Maps for Hex Meshing

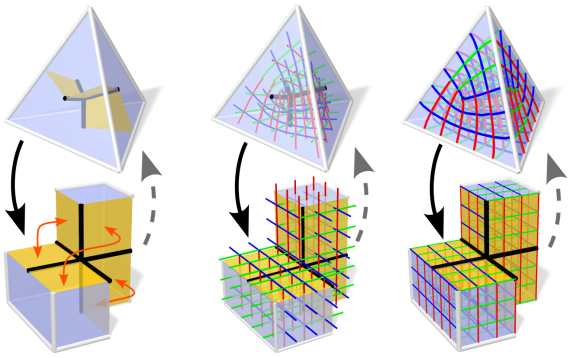
3

II …determine cut structure…

III Compute a seamless map.

IV Quantize it into an integer-grid map.

V Extract a hex mesh from it.



In the previous chapter we have introduced volumetric *polycube maps* as a feasible ingredient to hexahedral mesh generation. These polycube maps offer the advantage of a powerful reformulation of the hex meshing problem in terms of maps, which can be continuously optimized by established numerical methods. Intuitively, polycube maps are maps that deform an input volume such that it covers exactly the cells of a grid in parameter space. Inverse deformation of the grid cells then pulls them back into the original input, implying a hex mesh in the domain. Rather than specifying a grid explicitly, it is convenient to instead assume a canonical grid, namely the *integer-grid*, i.e., the Cartesian grid formed by all points with integer (parametric) coordinates.

Polycube maps can be summarized as deforming the input volume, such that its boundaries lie on the integer-grid. As has been recognized in the previous chapter, this is not truly general; there are hex meshes that are unreachable via polycube maps, specifically those with internal singularities. Around interior singular edges, the mesh connectivity does not resemble a regular grid, making it impossible for a deformed grid to represent this configuration.

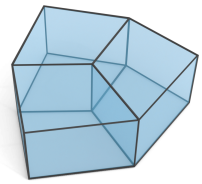
To be able to represent all possible hex meshes, a more general class of maps is required. Conceptually, it must be possible to *cut* the input volume along certain surfaces prior to deforming it onto the integer-grid. Equivalently, one must introduce the possibility for discontinuities in the map. These discontinuities across cuts, however, may not be arbitrary; rather, the two sides of a cut must be linked by certain *transitions*. As long as the two sides of a cut are constrained to overlay the integer grid in a symmetrically equivalent way, the grid's preimage "looks" continuous across cuts in the original domain, and therefore represents a valid hex mesh.

Integer-grid maps are essentially a generalization of polycube maps, still requiring alignment of the map's image with the integer-grid, but additionally allowing for map discontinuities, as long as these are linked by grid-preserving transitions. In this sense, they share some of the advantages of polycube maps (a compactified search space, map optimization as proxy for hex quality optimization), while being truly general, i.e., able to represent all possible hex meshes.

3.1 An Intuitive View

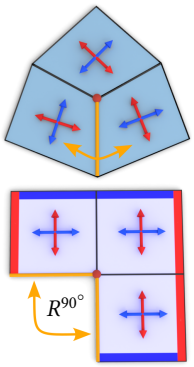
Before we begin with technical definitions, let us establish a more intuitive understanding of cuts, why they are necessary, and which transitions are admissible. To this end, we go the reverse way, starting from hex meshes that cannot be directly deformed onto the integer-grid and considering how cutting them allows to do so. Once this is understood, we can generalize the process to arbitrary inputs (rather than just hex meshes) and define integer-grid maps on these.

The simplest example of a hex mesh that requires cuts to be deformed onto the integer-grid is one consisting of three hexahedra surrounding an interior singular edge, as shown in the inset. Essentially, it is the extrusion of three planar quadrilaterals surrounding a point singularity. Let us therefore begin by considering such situations in 2D.



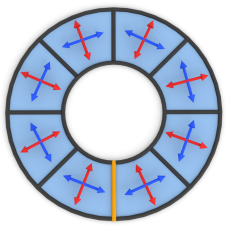
3.1.1 In Two Dimensions

The crux of being unable to deform a simple quad mesh like the one in the inset onto the integer grid, such that each quad covers a square grid cell, clearly is the interior singularity; around it, the mesh does not resemble a regular grid. A workaround is obvious, though: Cutting the mesh along any edge connecting the boundary to the singularity (orange in the top inset) allows to continuously deform it onto the grid afterwards (bottom in the inset). Notably, if we try to assign globally consistent quad orientations as crosses—representing the two principle axes each quad should be aligned with—a discontinuity of cross orientations appears at the cut as well. Across the cut, the two resulting sides (orange) are not truly disconnected; to encode a conforming quad mesh, their images under the parametrization must agree—up to transformation by a certain *transition* (orange arrow). In the above example the transition must perform a 90° rotation (and a shift) to transform one cut side into the other. The cross orientations across the cut are also linked via a 90° rotation.

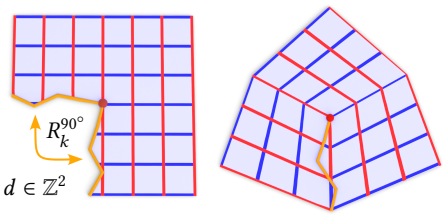


Preserving a conforming quad mesh structure across cuts implies that only rotations preserving the intrinsic symmetry of the grid-aligned, square quadrilaterals may occur at a cut. This is identical to demanding that the rotational part of transitions is grid-preserving, i.e., it maps the (infinite) 2D integer grid onto itself (swapping inessential axis labels at most). In 2D this is fulfilled by rotations $R_k^{90°}$ of k-times multiples of 90°. The rotational part of cuts and their transitions, as demonstrated on the above example, can be determined via a field representing only quad *orientations* (not sizes, not concrete positions), called a *cross field* in its simplest form [Vaxman et al. 2016]. The integer-grid (due to being infinite), however, also has translational symmetries; shifting it by any 2D integer vector d transforms the grid into itself. In summary, admissible transitions across integer-grid map discontinuities in 2D are the *automorphisms* of the 2D integer grid, composed of rotations $R_k^{90°}$ and shifts $d \in \mathbb{Z}^2$.

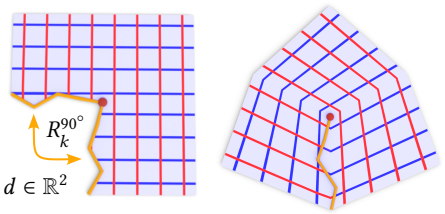
Beside enabling interior singularities, cuts can also serve to turn a given 2D shape into a topological disk, as shown in the inset for an annulus. Note that here, there is no rotation across the cut; the quad strip, despite being curved in object space would become straight in parameter space. The fact that integer-grid maps can not only model such simple examples, but *any* quad mesh, is straightforward to demonstrate: Take an arbitrary quad mesh, cut it along each quad edge, and map each quad individually onto an integer-grid square. Any potential transitions between grid squares are obviously grid-preserving, so the constructed map is indeed an integer-grid map, and the pre-image of the integer grid under the map recovers the



exact same quad mesh.



Rather than just cutting, mapping and reconstructing quad meshes, we can inject an arbitrarily represented 2D shape into the process. Cutting that shape—with consistent linking of cut sides—and mapping it onto integer-grid cells such that each cell is fully covered, guarantees that the grid's preimage under the map produces a boundary-aligned quad mesh of the shape. Actually, this notion is a bit too restrictive, as it requires cuts to be aligned with integer-grid lines. The inset demonstrates that this is not actually necessary; simply demanding that singularities and boundaries are aligned with the integer grid suffices.



Actually, this view presents a straightforward continuous relaxation of the mapping problem: Allowing non-integer coordinates for boundaries and singularities opens up a larger class of maps, so called *seamless maps*. For these, the rotational part of transitions is still required to be grid-preserving but the translational part is relaxed to arbitrary vectors $d \in \mathbb{R}^2$. While the more general seamless maps do not imply a quad-only mesh by pullback of the integer grid, as demonstrated in the inset, they retain some essential properties of integer-grid maps, which makes them valuable assets in quad mesh generation [Campen et al. 2015].

The relations between discontinuities and singularities of a surface cross field, of a seamless surface map, and of a quad mesh are summarized in fig. 3.1.

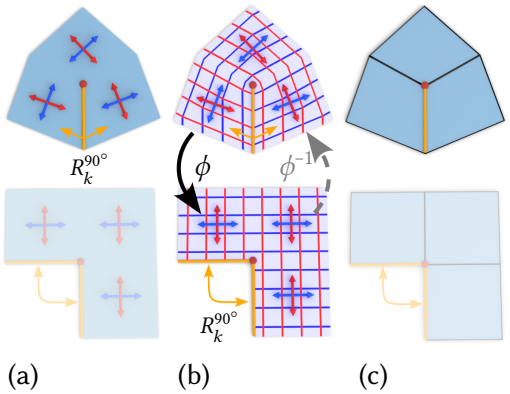


Figure 3.1: Equivalent discontinuities and singularities (a) of a cross field (top), with the implied map topology (bottom); (b) of a seamless map ϕ (top: object space, bottom: parameter space); (c) of a quad mesh (top), with the implied map topology (bottom). For computation of seamless surface maps, cross fields are typically used to pre-determine singularities and to guide parameter gradients, as indicated in the center. Crosses and map coordinates across discontinuities are linked via equivalent rotations $R_k^{90°}$ by multiples of 90°.

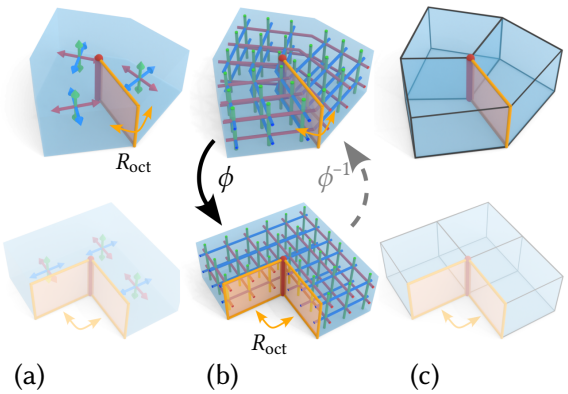
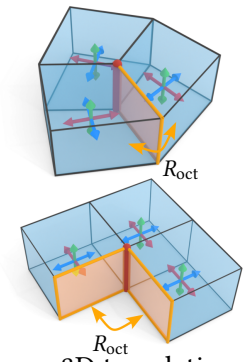


Figure 3.2: Equivalent discontinuities and singularities (a) of a frame field (top), with the implied map topology (bottom); (b) of a seamless map ϕ (top: object space, bottom: parameter space); (c) of a hex mesh (top), with the implied map topology (bottom). For computation of seamless volume maps, cross fields are typically used to pre-determine singularities and to guide parameter gradients. Frames and map coordinates across discontinuities are linked via equivalent rotations R_{oct} from the octahedral symmetry group.

3.1.2 In Three Dimensions



Conceptually, the picture does not change much when going to three-dimensional integer-grid maps: A 3D integer-grid map of a volume is a combination of cutting it and mapping it onto a 3D regular grid. However, the fact that all relevant entities are of increased dimensionality in this setting, makes things more complicated. The domain boundary is a surface rather than a curve, singularities are no longer isolated points but networks of curves, and cuts are no longer curves but form surfaces. Hence, conditions are now imposed on the correct linkage between the two sides of a cut surface. Just as in the 2D case, the conditions are that the transition from one side to the other must be *grid-preserving*. For the 3D integer grid, this means that the transition must be composed of an integer 3D translation $d \in \mathbb{R}^3$ and a rotation R_{oct} from the sign-preserving octahedral group, i.e., the symmetry group of an octahedron or, equivalently, of a cube.

Similar to quad orientations, which can be represented as crosses with a 4-fold rotation symmetry on surfaces, 3D frames (of implicit octahedral symmetry) serve as the foundation for cube orientation fields in 3D. The relations between discontinuities and singularities of a volumetric frame field, of a seamless volume map, and of a hex mesh are summarized in fig. 3.2. In the following we establish a formal notation for both seamless and integer-grid maps which is employed as a baseline throughout subsequent chapters.

3.2 Formalization

We assume in the following to be working on a tetrahedrized version of the original input domain. All the below notions could be generalized to abstract representations of 3-manifolds, but due to virtually all existing methods employing tet meshes in the background, a notation based on tet meshes more directly links to implementations. As defined in section 2.2, a tet mesh is a combinatorial polyhedral complex $\mathcal{M} = T \cup F \cup E \cup V$ consisting of tetrahedra $t \in T$, triangular **f**acets $f \in F$, **e**dges $e \in E$, and **v**ertices $v \in V$. Its connectivity is expressed by top-down and bottom-up incidence operators ∂ and ∂^{-1}, as well as expressions derived from these. Directed versions of facets and edges are denoted as half-facets $\vec{f}$ and half-edges $\vec{e}$.

The integer grid We refer to the integer grid in $\mathbb{R}^3$ as the set of all points in parameter space with at least one integer Cartesian coordinate:

$$G = \left\{(u_1, u_2, u_3) \in \mathbb{R}^3 \mid u_1 \in \mathbb{Z} \vee u_2 \in \mathbb{Z} \vee u_3 \in \mathbb{Z}\right\} \tag{3.1}$$

The grid is formed by integer isoplanes, having one common coordinate that is integer. Integer isolines are formed as intersection of two orthogonal integer isoplanes, and integer points as intersections of three orthogonal integer isoplanes.

Chart-based map In this context, we consider a chart-based map ϕ of a tetrahedral mesh to be given by a collection of maps and their domains $\{(T_i, \phi_i)\}$—commonly called an atlas of charts—induced by a complete partition of tetrahedra $T = \bigcup_i T_i$. Each map $\phi_i : T_i \to \mathbb{R}^3$ is defined only on the associated subset of tetrahedra T_i and maps them into $\mathbb{R}^3$. We assume each ϕ_i to be represented in a piecewise linear manner as per-vertex parameters $\boldsymbol{u}_v$. Then the restriction of the map $\phi_i|_t : t \to \mathbb{R}^3$ to a tet t, represented per-vertex as $\boldsymbol{u}_v|_t$, is affine. Maps of similar type are often referred to as a *parametrizations* of the input domain; we use these terms synonymously.

The maps ϕ_i and their domains are necessarily overlapping; even if all T_i, T_j are pairwise disjoint they still share elements via their boundary. Maps from different charts are required to agree in such overlaps, but the notion of "agreement" is flexible. For integer-grid maps it is based on the infinite parametric grid G; in a slight abuse of notation, agreement requires that $\phi_i^{-1}(G) = \phi_j^{-1}(G)$ on any overlaps, i.e., the inverse map must exactly preserve the grid across charts.

Each individual map is intuitively required to act like a polycube map, aligning the mesh boundary with integer isoplanes. Implementation-wise, a chart-based map on a tetrahedral mesh is easily modelled by assigning a chart (t, ϕ_t) per tet, i.e., assuming the maximum disjoint partition. This will also be the default notion used in the following.

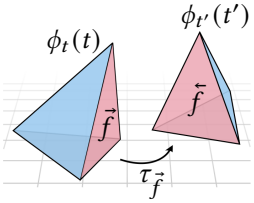
Angle defect The local image $\phi_t(t)$ of each tet t forms a certain parametric dihedral angle $\theta^\phi_{t,e}$ at each of its edges e. Hence, the total parametric angle for any edge can be computed as $\Theta^\phi_e = \sum_{t \in \partial^{-2} e} \theta^\phi_{t,e}$. We define the angle defect of edge e as

$$\Omega^\phi_e = \begin{cases} \pi - \Theta^\phi_e & \text{if e is a boundary edge} \\ 2\pi - \Theta^\phi_e & \text{otherwise} \end{cases} \tag{3.2}$$

Definition 3.1 (Integer-grid map). An integer-grid map is a chart-based map ϕ of a tet mesh $\mathcal{M}$, consisting of one chart (t, ϕ_t) per tet $t \in T$ and represented as parametric vertex coordinates $\boldsymbol{u}_{v,t}$ within each chart, such that [Nieser et al. 2011]:

IGM$_\mathbf{I}$ ϕ is locally injective, i.e., the determinant of the Jacobian matrix $\det J_t$ for each ϕ_t is positive.

IGM$_\mathbf{II}$ The affine transitions between charts $\tau_{\vec{f}} = \phi_{t'} \circ \phi_t^{-1}$, observed when passing from the chart of one tet t to that of an adjacent tet t' via the half-facet $\vec{f}$, are automorphisms of the integer grid:



$$\begin{aligned} \forall \vec{f}\ \forall v \in \partial^2 \vec{f}: \quad \boldsymbol{u}_{v,t'} &= \tau_{\vec{f}}(\boldsymbol{u}_{v,t}) \\ &= R_{\vec{f}} \boldsymbol{u}_{v,t} + d_{\vec{f}} \qquad \text{with } R_{\vec{f}} \in R_{\text{Oct}}, d_{\vec{f}} \in \mathbb{Z}^3. \end{aligned} \tag{3.3}$$

Here R_{oct} are rotations from the chiral octahedral group, i.e., $R_{\vec{f}}$ must correspond to a signed permutation of the Cartesian axes.

IGM$_\mathbf{III}$ Boundary facets of $\mathcal{M}$ are mapped onto parametric isoplanes with integer coordinates.

IGM$_\mathbf{IV}$ Edges of $\mathcal{M}$ that are singular under ϕ, identified by a nonzero angle defect, are mapped onto parametric isolines with integer coordinates.

It follows from this definition that the problem of finding integer-grid maps requires choosing integer values for transition translations due to **IGM$_\mathbf{II}$**, as well as for the aligned coordinates of boundaries and singularities due to **IGM$_\mathbf{III}$** and **IGM$_\mathbf{IV}$**. Relaxing these integrality constraints opens up the larger class of seamless maps, defined below. For clarity, when we refer to a seamless map, we always mean one that is *not* also an integer-grid map, unless explicitly stated otherwise.

Definition 3.2 (Seamless map). A seamless map is the continuous relaxation of an integer-grid map, for which the integer translations $d_{\vec{f}}$ of transitions as well as the aligned coordinates of boundary facets and singular edges may take on arbitrary rather than integer values. Otherwise, all conditions of theorem 3.1 remain in effect.

3.3 Properties

Map singularities The criterion for edges being singular under ϕ is given above: A non-zero angle defect $\Omega_e^\phi \neq 0$. In fact, for seamless and integer-grid maps, Ω_e^ϕ is always $i \cdot 90°$, i.e., an integer multiple of 90° [Nieser et al. 2011]. Several notions of a singularity index or valence, based on i are employed in literature. To differentiate it from fractional indices of frame field singularities or strictly positive hex mesh singularity valences, we denote map singularity indices directly via i. The valence v of the resulting hex mesh singularity is given as $v = 4 + i$, as each 90° sector in the map fits exactly one cubic grid cell. Just like for hexahedral meshes, non-singular edges, as well as vertices with no incident singular edges, are referred to as *regular*. Similarly, we refer to maximum chains of singular edges as *singular links* and to the vertices where links end or branch as *singular nodes*. For both integer-grid and seamless maps, the network of singular links and nodes directly corresponds to the singular graph of the implied hex mesh, warranting the same naming convention. Roughly speaking, we can say that the *topology* of an integer-grid map ϕ on a tet mesh $\mathcal{M}$ is given by the topology of its singular graph embedded within $\mathcal{M}$ and the behaviour of ϕ along topological features of $\mathcal{M}$. In that way, an integer-grid map and a seamless map can have the same topology.

Index limits It is common that works limit singularity indices to ones typically obtained from reasonable frame fields; these rarely go above $i = 2$ or below $i = -1$. In this case, the characterization of singularities can be replaced by a different notion, namely that the composition of transition functions on a shortest cycle around them is not the identity, i.e., $\bigcirc_{\vec{f} \in \partial^{-1}\vec{e}} \tau_{\vec{f}} \neq I$. Intuitively, this means that the transition function must bridge the gap to complete a full 360° rotation. This fails to capture only singularities with an index i that is integer-divisible by 4—excluded by the assumed index limits. Furthermore, the isoline alignment part of condition **IGM$_{\text{IV}}$** is required only for singularities with index 4, 8, ...For any other singularities, a non-identity cyclic transition around the singularity already implies that singularities must consist of fixpoints under this composed transition, which for 1D curves implies alignment with a parametric isoline (see the case analysis in section 4.2.3.5). Note, that the integrality part of constraint **IGM$_{\text{IV}}$** is still necessary for any singularity type, including those of multiple-of-4 index.

Cut surface We call each facet f for which $\tau_f \neq I$ a *cut facet*, and refer to the entirety of cut facets as the *cut surface*. The cut surface of the map can be arbitrarily dense, effectively separating each adjacent per-tet chart in parameter space. However, for any such integer-grid map, one that implies the same hex mesh but has a topologically simple cut surface exists, i.e., one that cuts the domain exactly into a single topological ball [Jiang et al. 2014]. W.l.o.g. we assume that a cut surface in minimal form is given; this is not hard to achieve algorithmically [Bommes et al. 2009]. A minimal cut surface contains full cuts that transform the domain into a topological ball, and partial cuts that lay bare interior singularities (except for previously mentioned multiple-of-4 singularities). That means that on the imagined domain resulting from actually applying the virtual cuts, all singularities would lie within the boundary—just like for a polycube. Hence, the integration of such virtual cuts is exactly the mechanism via which integer-grid maps generalize polycube maps. Furthermore, the presence of interior singularities of positive index—surrounded in parameter space by a total parametric dihedral angle of more than 360° and laid bare on the boundary by a partial (virtual) cut of a minimal cut surface—implies that the image of $\mathcal{M}$ under ϕ may overlap itself if a minimal cut surface is chosen. This is indeed allowed for seamless and integer-grid maps; the requirement of injectivity applies only on a local scope.

Injectivity & Quality The map's Jacobian matrix J_t within tet t with vertex indices i, j, k, l, per-vertex object space coordinates $\boldsymbol{x}$ and parametric coordinates $\boldsymbol{u}$ is given as:

$$J_t = \begin{bmatrix} | & | & | \\ \boldsymbol{u}_j - \boldsymbol{u}_i & \boldsymbol{u}_k - \boldsymbol{u}_i & \boldsymbol{u}_l - \boldsymbol{u}_i \\ | & | & | \end{bmatrix} \begin{bmatrix} | & | & | \\ \boldsymbol{x}_j - \boldsymbol{x}_i & \boldsymbol{x}_k - \boldsymbol{x}_i & \boldsymbol{x}_l - \boldsymbol{x}_i \\ | & | & | \end{bmatrix}^{-1} . \tag{3.4}$$

As mentioned, a locally injective seamless or integer-grid map requires $\det J_t > 0$ everywhere. This property also guarantees that there exists a hexahedral mesh with only valid cells, extractable from the integer-grid map. However, there is no guarantee that a mesh obtained from connecting the preimage of the integer-grid corners via *linear elements* is geometrically valid everywhere; potentially, curved elements or a denser parametric grid (e.g., including half-integer coordinates) are needed. In general, $\det J_t$ is a measure for volume distortion of the map, which directly translates to equivalent distortion of cells in the final hex mesh. Hence, a combination of minimal volume distortion, indicated by values for $\det J_t$ close to 1, and minimal angular distortion, indicated by nearly equal singular values of J_t, is commonly chosen as the quality objective for integer-grid map generation.

Conformity & Accuracy Conformity to the input domain boundaries is explicitly ensured in the formulation of integer-grid maps via **IGM$_{\text{III}}$**. As the map's image is constrained to be integer everywhere along the domain boundary, integer-grid corners—representing later hex mesh vertices—exactly sample the input boundary with directly controllable density. Hence, by construction, boundary vertices of the output hexahedral mesh lie exactly on the input domain's boundary, and are connected by boundary-aligned quadrilateral facets that smoothly approximate the surface without staircase artifacts. The approximation error along curved boundaries, measurable for instance via the Hausdorff distance between the input and output boundaries, can therefore be predictably controlled via the map's scale. Applying a global, uniform scaling after map generation (equivalent to subdividing the integer-grid before hex mesh extraction) improves the approximation everywhere without affecting element distortion. When locally adaptive resolution (or anisotropy) is desired instead, a field of local scaling factors (or tensors) can be considered already during map generation [Xu et al. 2017; Fang et al. 2021]. Such a field modifies the metric under which each J_t is computed and thereby effectively trades some amount of geometric element quality and structural simplicity in the ouput for the desired resolution adaptivity.

Feature Handling The base formulation by construction ensures the conformity of output hexahedral mesh cells only with boundaries of the input. The goal for feature preservation additionally is that—in the final hex mesh—feature surfaces are closely approximated by sheets of quadrilaterals, feature curves by chains of edges, and feature points by vertices. Then, the principles discussed previously also hold for feature approximation accuracy. Achieving this simply requires additional constraints of type **IGM$_{\text{III}}$** for feature surfaces, curves and points to be mapped onto integer isoplanes, isolines and integer points, respectively. For seamless maps, being continuous relaxations, arbitrary non-integer isoplanes, isolines and points suffice. Additionally, to ensure the boundaries of features to also be preserved, it is sensible to assume:

- Feature curves are cyclic or bounded by feature points.
- Feature surfaces are closed or bounded by feature curves.

3.4 Hex Meshing via Integer-Grid Maps

As touched upon in the previous section, to obtain hex meshes satisfying the needs of validity, quality, and accuracy, the envisioned approach is a reformulation via the constrained mixed-integer optimization problem of generating low-distortion integer-grid maps. Mesh validity, as well as feature and boundary conformity are ensured via explicit constraints, element density and isotropy are controllable via the metric of the map, and structure and geometric quality are directly related to map distortion. Assuming an abstract energy function $E(\phi)$ that indirectly measures (un)desirable hex mesh properties via the integer-grid map ϕ, the problem becomes:

$$\min_{\boldsymbol{u}_v|_t, R_f, d_f} E(\phi) \quad \text{s.t. } \mathbf{IGM_I} \text{ to } \mathbf{IGM_{IV}}. \tag{3.5}$$

Tractability The problem, while now in a form better suited for numerical optimization than the direct hex meshing problem, is still highly complex. The main difficulty arises from the high number of discrete variables: The transition rotations R_f are constrained to be from the 24-element set of rotations of the octahedral symmetry group, and transition translations d_f as well as some parametric coordinates $\boldsymbol{u}$ of boundaries and singularities must be integer. As of today, no feasible method is known that aims to solve the problem eq. (3.5) via a single optimization procedure.

However, the reformulation of the hex meshing problem via eq. (3.5) has another key advantage: It offers a whole range of partial relaxations of the complete problem, by considering only subsets of constraints. This enables treatment in an algorithmic *pipeline*; each stage only solves the problem partially, but a sequence of partial solutions building on each other eventually yields a full solution. Disregarding the interdependency between these stages results in some degree of suboptimality of the obtained solution, but makes the problem tractable in practice. The four main stages of the pipeline, discussed in more detail in section 3.6, are orientation-field computation (fixing R_f and hence the map's topology), seamless map computation (optimizing continuously all d_f and $\boldsymbol{u}$), quantization (fixing d_f and some $\boldsymbol{u}$ to valid, near-optimal integers), and integer-grid map computation (optimizing continuous variables with all integer variables fixed).

3.5 Parallels to Quad Meshing

Integer-grid maps, both as a concept [Ray et al. 2006; Kälberer et al. 2007; Bommes et al. 2009] and as a term [Bommes et al. 2013], originate from seminal works in surface parametrization and quad remeshing. Most of the work on integer-grid-map hex meshing is built on the foundations laid in the surface domain, warranting a brief overview of the latter here. Solving the integer-grid map problem eq. (3.5) *directly* turns out to already be intractable in 2D with currently known methods [Coudert-Osmont et al. 2024]. A pipeline consisting of several steps, each fixing only some degrees of freedom, has therefore been established for the 2D case as well.

3.5.1 Orientation Fields

The first, most fundamental degrees of freedom to fix also in quad meshing are the rotational parts of transitions, and the singular structure they induce. These degrees of freedom are coupled in an intuitive way to the *orientation* of quads in the later quad mesh. Inversely, prescribing quad orientations—without specifying their sizing or concrete position—implies the singular structure of a map. This relation is illustrated in fig. 3.1. Providing some representation of orientation

everywhere on the domain amounts to computing an *orientation field*. Orientation fields for quad mesh generation have been described first for the 2D plane [Knupp 1995] and later lifted to surfaces [Ray et al. 2006]. Different representations for the rotation symmetry of quad orientation have been explored in multiple works, and reviewed by Vaxman et al. [2016]. Design of such fields is typically built around alignment to principal curvature directions and field smoothness.

3.5.2 Seamless Parametrization

In terms of the map, local quad orientations correspond to the gradient directions of the map's two parametric components. Hence, one often speaks of *integrating* an orientation field into a seamless map. In practice, this amounts to minimizing some form of alignment energy that measures discrepancy between parameter gradients and the prescribed orientation field directions. This process is more robust if the orientation field is *integrable*, i.e., curl-free—a property that requires explicit tailoring of field design [Diamanti et al. 2015; Coiffier and Corman 2023; Corman and Crane 2025; Couplet et al. 2025]. A related aspect is the *meshability* of a given orientation field, or more specifically of its topology, implied by its singularities and its behavior along certain loops on the surface. Meshability indicates whether a 2D seamless map—and by virtue a quad mesh—of equivalent topology exists. Conditions for meshability in absence of features or boundaries turn out to be relatively mild in 2D [Shen et al. 2022]; only rather specific, easily avoided types of non-meshable orientation fields exist. However, even if a topologically equivalent seamless map can be reliably found for most orientation fields, such a map might be arbitrarily distorted, due to being constrained to match the field topology. In practice, low-quality orientation fields can numerically challenge even recent approaches with theoretical guarantees of obtaining an injective map [Campen et al. 2019; Shen et al. 2022]. Hence, a "good" orientation field is crucial for obtaining a high-quality seamless map.

3.5.3 Quantization

Essentially, any method that computes an integer-grid map via mixed-integer optimization employs a mechanism to transform a continuous solution, i.e., a seamless map, into a quantized one, i.e., an integer-grid map. The issue with doing so in an implicit way, most commonly done in earlier approaches by naively rounding variables to the closest integer [Kälberer et al. 2007; Bommes et al. 2009, 2010], is that this might make the integer-grid map problem infeasible. Specifically, rounding can force points that are spatially distinct in object space to coincide in parameter space (accounting for transitions), which forcibly violates the injectivity condition $\mathbf{IGM_I}$ and leads to defects in the resulting quad mesh if processed further. While a backtracking algorithm has been proposed [Bommes et al. 2013], detecting the infeasibility of integer choices through failure to find an injective map, this approach is prohibitively slow for complex domains due to an unnecessarily complex problem having to be solved for each integer evaluation.

A step forward in terms of both robustness and efficiency was the explicit, separate determination of valid integer values. Campen et al. [2015] use a seamless map to construct a T-mesh on the input, essentially serving as the proxy domain for a simplified proxy quantization problem. The solution of this T-mesh quantization problem, an assignment of integer sizes to the T-mesh elements, can then exactly be translated back into valid and near-optimal integers for the respective integer-grid map problem. The coarse global connectivity of the T-mesh, combined with the explicit representation of critical points like singularities, can be exploited to guarantee that the chosen integers are valid, i.e., they admit an injective integer-grid map. The T-mesh surface partition the authors employ is constructed as a so called *motorcycle graph* [Eppstein et al. 2008]

of the seamless map. Similar to the construction routine of base complexes on structured meshes (see section 2.3.3.1), it is obtained by expanding surface-partitioning lines from singularities but in this case assumes the lines to be mutually blocking (see fig. 5.1(a) for an illustration). As an alternative coarse proxy domain, Coudert-Osmont et al. [2024] recently suggested a coarse (decimated) triangulation of the input.

3.5.4 Surface T-Meshes

Beside uses in surface parametrization [Myles et al. 2010] and quantization [Campen et al. 2015], surface T-meshes have been employed for various other purposes. Some works trace a T-mesh for quad meshing already on the cross field [Myles et al. 2014; Pietroni et al. 2016, 2021], implementing additional measures to ensure meshability of the resulting partition. Variants of the map-based motorcycle graph so far have been used to find guaranteed-injective initializations of integer-grid maps [Lyon et al. 2019], and in obtaining maximally coarse patch layouts of the input [Lyon et al. 2021a,b]. T-meshes built on slightly different inputs have been employed in T-spline construction [Campen and Zorin 2017; Karčiauskas et al. 2017].

3.6 The Integer-Grid Map Hex Meshing Pipeline

In the following, we will detail the (minimal) pipeline for hexahedral meshing with integer-grid maps, as established in prior literature. This pipeline, in large parts, is a direct extension of the quad meshing pipeline discussed above, and requires solving similar sub-steps. Specifically, we identify robustness and quality gaps in the prior state-of-the-art pipeline that motivate the approach taken in this thesis and the contributions developed in its scope.

Input The input is a tet mesh $\mathcal{M} = T_\mathcal{M} \cup F_\mathcal{M} \cup E_\mathcal{M} \cup V_\mathcal{M}$ as defined in section 2.2. Optionally, feature surfaces, curves and points may be supplied as marked subsets of facets, edges and vertices $\hat{F}, \hat{E}, \hat{V}$. This corresponds to the formatting of the recent HEXME dataset [Beaufort et al. 2021], designed specifically as a benchmark for hex meshing approaches. The target resolution of hex elements may be specified as a target edge length, target element count or more generally a target sizing field.

Output The output is a hex mesh $\mathcal{H} = H_\mathcal{H} \cup F_\mathcal{H} \cup E_\mathcal{H} \cup V_\mathcal{H}$ as defined in section 2.3, matching the desired resolution as closely as possible and faithfully recovering the shape of input boundaries and features. In $\mathcal{H}$, input feature surfaces, curves and points must be represented as sheets of quadrilaterals, chains of edges or vertices, respectively.

Pipeline stages A seminal work in hex meshing with integer-grid maps was CUBECOVER developed by Nieser et al. [2011], lifting many of the concepts previously developed for surfaces [Kälberer et al. 2007] to the volumetric domain. The CUBECOVER pipeline was adopted by most subsequent approaches, and encompasses the following steps:

Orientation field design Computation of an orientation field on the tet mesh, constrained to align with boundaries and features. This fixes the rotational part of the problem, specifically the axis-permuting rotations $R_{\vec{f}}$ between charts, and implies the singular graph for the remaining steps.

Field-aligned integer-grid map Computation of an integer-grid map on the tet mesh via mixed-integer optimization of eq. (3.5) but with $R_{\tilde{f}}$ fixed. Integer constraints are satisfied by direct rounding of intermediate continuous solutions.

Hex extraction Extracting the hex mesh as the preimage of the integer grid under the final map.

3.6.1 Frame Fields

In 3D, orientation fields representing hexahedra are typically referred to in the most general sense as *frame* fields, implying their relation to (symmetrized) 3-vector coordinate frames. A 3D integer-grid map and a corresponding field of coordinate frames are directly linked; given an integer-grid map, for every point in the domain a frame is easily obtained by combining the map's component-wise gradient vectors. Obtaining a frame field that corresponds to feasible parametric gradients *without* prior knowledge about the implied parametrization, however, is an intricate problem—even more so than in the surface domain discussed before. A majority of works in the field therefore explicitly target the frame-field design stage, proposing different frame representations, different field generation algorithms, and different field correction techniques, all to cope with issues of non-integrability and non-meshability.

Frame field representation

The intuitive representation of frame fields via 3-tuples of linearly independent vectors (or equivalently, a matrix formed by these vectors) [Nieser et al. 2011] directly translates to the three axes of a linearly deformed cube and can be seen as a generalization of vector fields. Importantly, though, frame fields always imply a symmetrization by reflection of the three vectors across the origin, together pointing to the six corners of a linearly deformed octahedron, i.e., the cube's topological dual. Furthermore, the three vector pairs are considered indistinguishable; any sign-preserving permutation of the vector-pair tuple represents the same frame. Hence, there exist 24 different 3-tuples of vectors that are identified with the same frame via equivalence relations [Nieser et al. 2011]. This induces a topological structure, including singularities, that is much more complex than that of vector fields. Nieser et al. [2011] argue, in an extension of the equivalent 2D concept [Kälberer et al. 2007], that frame fields are equivalent to vector fields on a 24-sheeted branched covering of the domain, although such a representation has not yet actually been employed in practice.

The space of frames representable by 3-tuples of vectors is commonly restricted further, via *odeco frames* implying orthogonality or *octahedral frames* implying orthonormality. The latter, due to being more constrained, can be much more compactly encoded than via 3-vector-tuples: Both unit quaternions [Gao et al. 2017a; Liu et al. 2018] and Euler angles [Huang et al. 2011; Ray et al. 2016; Palmer et al. 2020] have been used for frame parametrization. Generalizing these encodings to odeco frames in addition only requires three scaling factors. The crux of representations, though, is to find a way of expressing the 24-fold symmetry equivalence that admits a computationally feasible field optimization algorithm. Using the 3-vector-tuple representation requires an additional integer variable for each pair of neighboring frames that represents one of the 24 possible (signed) axis permutations that correspond to rotations from the octahedral group OCT. These integers and the corresponding permutations match (signed) axes of neighboring frames are thus called *matchings*, and directly correspond to $R_{\tilde{f}}$ in eq. (3.5). To avoid explicitly handling integer variables in the optimization problem, representations can be used

that are already invariant under OCT, recovering the implied matchings only after optimization. OCT-invariant representations are commonly based on polynomials in an inherently symmetrical basis, such as spherical harmonics [Huang et al. 2011; Ray et al. 2016; Desobry et al. 2021], or on a symmetry-restricted class of higher-order tensors [Chemin et al. 2018; Palmer et al. 2020; Golovaty et al. 2021; Vekhter et al. 2025]. Both types of approaches lift frame representations into a much higher-dimensional but constrained space, replacing the explicit handling of integer matchings in the optimization procedure by high-order polynomial functions as objective and equally non-linear constraints, all coming with their own practical challenges.

Another interesting approach to avoid integer matchings is to express frames not explicitly but via local derivatives, i.e., changes between neighboring frames, so that frames can be recovered by specifiying just a single frame orientation and reconstructing the rest as prescribed by the differentials—via discrete integration, so to speak. Corman and Crane [2019] propose such an approach that can guarantee path-independence of the reconstruction process, but inputs are restricted to certain shapes and the singular graph is assumed given in advance.

Frame field optimization

Finding a frame field suited as a precursor to eq. (3.5) is once more an optimization problem of minimizing an energy expressed in the frame representation, constrained to align frames with both domain boundary and features. As opposed to the surface domain, volumetric 3D space is not curved, hence measures like principal curvature alignment can not be used as frame field guidance. Instead, the energy is typically chosen to represent frame field *smoothness* in some way, because this most directly translates to low map distortion. Smoothness of a (discrete) function is commonly expressed via the (discrete) Dirichlet energy, i.e., the squared norm of the field's gradient. Another important aspect that can already be accounted for on the frame-field level is that of element sizing and anisotropy. Typically, these aspects are decoupled from the frame-field optimization in a pre-process that first computes a precursor, which acts as a metric field replacing the default metric in the frame-field optimization [Xu et al. 2017; Fang et al. 2021].

Research has shown that already the discretization (tet-based [Liu et al. 2018], facet-based [Huang et al. 2011], vertex-based [Ray et al. 2016; Palmer et al. 2020], or boundary-only [Solomon et al. 2017]) and the numerical frame representation on which gradients are computed (see above) directly affect the optimization and its results. Specifically, the performance of optimization schemes heavily depends on the above choices. For representations lifted to higher-dimensional spaces but constrained to non-trivial sub-manifolds within these, measures are required to enforce these constraints. Concretely, operators projecting intermediate solutions onto the feasible set [Huang et al. 2011; Ray et al. 2016; Palmer et al. 2020] are required. An increase in performance—by increasing the chances of avoiding local minima—has been shown when interleaving diffusion steps with the core optimization [Palmer et al. 2020]. This has been conjectured, but not proven, to be at least similar (if not equivalent) to optimizing a Ginzburg-Landau type energy that includes a progressively upscaled penalty term, which punishes deviation from the feasible sub-manifold but makes the problem increasingly non-convex [Pietroni et al. 2022].

Frame field correction

The singular graphs of frame fields are general enough to cover all possible singular graphs of integer-grid maps and, by virtue, of hex meshes. However, the space of frame field singularities is actually larger than that of hex meshes [Viertel et al. 2016; Liu et al. 2018; Liu and Bommes 2023], meaning that some frame fields are not *meshable*; there simply is no translation between the frame field and a corresponding hex mesh, because no such hex mesh exists. The topic of meshability

in recent works is based completely off the singular graph. The most common approach is to compute an initial frame field, extract the singular graph from it, apply modifications aimed at making it meshable, and computing another frame-field constrained to match the modified singular graph [Li et al. 2012; Jiang et al. 2014; Liu and Bommes 2023]. However, while today such approaches are able to resolve local configurations that prevent meshability [Liu and Bommes 2023], the matter of global meshability proves to be much more intricate than in the 2D setting. No theory that fully explains or algorithm that fully establishes global meshability of frame fields has been found yet, constituting one major robustness gap that contributes to the high failure rate of current integer-grid map pipelines.

3.6.2 Integer-Grid Parametrization

Having extracted a singular graph and rotational matchings $R_{\tilde{f}}$ from a given frame field, virtually all recent methods attempt to directly solve the remaining mixed-integer problem via eq. (3.5), and—for lack of a better solution—resort to the naive rounding of continuous intermediate solutions to satisfy integer constraints, as proposed in early works on quad meshing [Kälberer et al. 2007; Bommes et al. 2009] and hex meshing [Nieser et al. 2011].

Even provided that this heuristic procedure of fixing integers succeeds and admits an injective integer-grid map, no approach can currently guarantee finding such a map [Pietroni et al. 2022], because the continuous problem that remains after all integers have been fixed still is highly non-convex (failures are demonstrated in figs. 9.18 to 9.20). While for the 2D case multiple works can guarantee finding injective maps even for problems with seamlessness and alignment constraints via an intrinsic convex [Campen et al. 2019, 2021; Shen et al. 2022] or combinatorial [Zhou et al. 2020; Levi 2022, 2023] reformulation of the problem, no such technique exists so far for the volumetric case. Context-agnostic numerical methods [Du et al. 2020; Garanzha et al. 2021], even though advertising practical success rates of (close to) 100% on some datasets, still often fail to obtain an injective map for general parametrization problems. A first attempt to generalize the first class of intrinsic reformulation approaches to 3D has been demonstrated [Paillé et al. 2015], but does not result in a convex problem. A few provably robust, combinatorial approaches to 3D parametrization have recently been introduced [Campen et al. 2016; Hinderink and Campen 2023; Nigolian et al. 2023; Hinderink et al. 2024; Nigolian et al. 2024], but are currently still limited in terms of generality and efficiency [Meloni et al. 2024].

Rounding The main problem with using variable-wise rounding to obtain an integer solution from a continuous one is the following: Imagine two distinct feature points, required to lie on integer parametric coordinates and independently rounded to the same integer (corrected for transitions); the parameter space covered by the mesh between these points must then have zero volume, i.e., the map must degenerate. The same may occur for opposite parts of the boundary, separate singular curves or other feature entities. Trying to extract a hex mesh from such a degenerate map causes irrepairable defects in the form of holes or missing topological features. Such failures are common and easy to provoke (see figs. 6.4 and 8.2 for concrete examples).

One may argue, as sometimes done in the literature, that this effect can be somewhat mitigated by using a sufficiently upscaled seamless map to begin with, increasing parametric gaps so that collapse under rounding is less likely. However, beside this still being an unreliable heuristic—failing for small enough features—it promotes a worse resulting mesh structure, due to singularities and features being more likely to be misaligned. Furthermore, it requires post-processing to reach the *actually* desired hex mesh resolution (which might be lower), and also compromises

solution optimality as the rounding disregards interdependencies of integer variables in both the constraints and the objective function.

3.6.3 Hex Extraction

The last remaining step in the hex meshing pipeline, namely extraction of a combinatorially valid hex mesh from the integer-grid map, is relatively straightforward and has been solved robustly. The original approach even offers some heuristics to correct numerically faulty inputs [Lyon et al. 2016]. Alas, large-scale map defects like such caused by non-meshable frame field singularities or integer rounding gone wrong can not be repaired via such heuristics. Hence, Kohler et al. [2025] recently published a much more efficient version of hex extraction that comes without such heuristics, arguing that the few cases where they actually would help are nowadays better handled in prior steps of the pipeline.

3.7 Contribution: Volume Parametrization Quantization

Parametrization quantization refers to the process of transforming a continuous seamless map into an integer-grid map implying a discrete hex mesh. A robust and efficient concept for quantization has been introduced in the 2D setting about a decade ago [Campen et al. 2015] and steadily refined in later works [Lyon et al. 2019, 2021a,b]. Yet, for hex meshing no generalization has been available ever since, explaining the persistent usage of a fragile heuristic based on rounding in recent works in the field. This robustness gap has remained unbridged for so long due to several aspects of the 2D problem and its solutions not generalizing easily to the volumetric domain.

Specifically, the following open problems have to be solved to establish a robust volume parametrization quantization that provides the same guarantees as the 2D pipeline:

- How to generalize the surface T-meshes that were the central tool to achieve quantization robustness to the volume? [addressed in chapters 5 and 6]
- How to construct such T-meshes? [addressed in chapters 4 and 5]
- How to solve the proxy quantization problem on volumetric T-meshes while ensuring validity? [addressed in chapter 7]
- How to translate the solution of the proxy problem to integer values for the integer-grid map? [addressed in chapter 8]
- How to find a guaranteed-injective integer-grid map with these values fixed? [addressed in chapter 9]

In parts B and C we successively answer these questions by providing constructive algorithmic solutions. Altogether these solutions establish a fully robust volume parametrization quantization. Building on this robust foundation, part D then explores two contributions aimed at improving practical utility of the volume quantization method, namely:

- A framework for adjusting the singularity graph during quantization, potentially simplifying the one prescribed by a precursor frame field, thereby yielding a block structure and mesh resolution better matching the desired target. [addressed in chapter 10]
- A tailored solver for the proxy quantization problem, improving implementation efficiency and flexibility by replacing black-box integer solvers that exhibit problematic worst-case behaviour (and potentially costly or restrictive licensing). [addressed in chapter 11]

By virtue of our contributions, the modified state-of-the-art pipeline for hex meshing via integer-grid maps, as outlined in section 1.5.2, becomes:

I Tet mesh the domain.

II Compute a meshable frame field on the tet mesh, fixing rotations of transitions.

III Compute a continuous seamless map with fixed rotations, without integrality constraints.

IV Quantize the seamless map into an integer-grid map.

- **IVa** Correct numerical issues in the seamless map, making it *truly seamless* .
- **IVb** Construct a coarse T-mesh decomposition, via the *3D motorcycle complex* algorithm.
- **IVc** Compute a valid proxy T-mesh quantization by assigning integer lengths to it.
- **IVd** Extract linear constraints from the quantized T-mesh.
- **IVe** Collapse zero-cells from the T-mesh.
- **IVf** Initialize a quantized, injective map per block; optimize it globally, s.t. **IVd** .

V Extract a hex mesh by pullback of the integer grid along the integer-grid map.

An illustration of the main stages of our novel volumetric quantization is presented in fig. 3.3. References to the current progress along this pipeline are given in each subsequent chapter's heading for quick orientation.

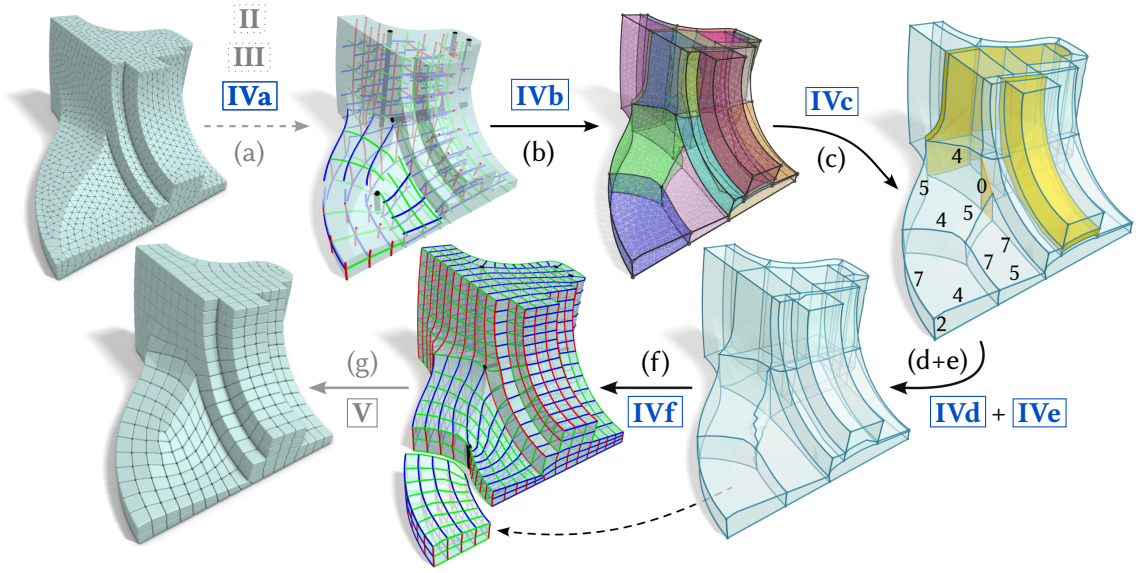


Figure 3.3: Our proposed robust quantization pipeline for hexahedral meshing. Given a tetrahedral mesh, (a) a truly seamless map ϕ is computed. (b) Using ϕ as a guiding structure, we construct a block decomposition of the volume, such that each block is an aligned cuboid under ϕ. (c) On the resulting partition, modeled as a volume T-mesh, we assign integer parametric target sizes to T-mesh cells, and (d) translate these into constraints for later map recomputation. (e) Any undesirable T-mesh cells, identified by an assigned size of zero (orange), are removed from the T-mesh through specialized collapse operators. (f) On the resulting partition with strictly positive integer sizes, an injective integer-grid map is initialized per block, with integer variables fixed according to step (d), and then globally optimized for. (g) Finally, the hex mesh is extracted as the preimage of the integer grid. With the contributions presented in this thesis, addressing parts of step (a) as well as steps (b) through (e), the problem of obtaining a discrete hexahedral mesh from a continuous seamless map can be considered robustly solved. On the remaining substeps of (a), progress in terms of robustness is ongoing [Liu and Bommes 2023].

Part B

T-meshes as Proxy Domains

My story begins at the dawn of time in the far away realm of Alphabetrium.

Ice-T reveals his true form
in *Rick and Morty: Get Schwifty*

4 Input Map Sanitization

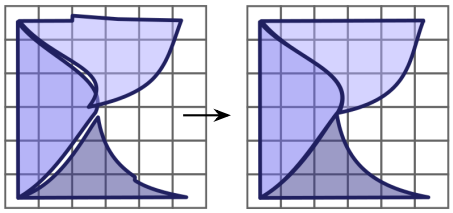
III Compute a volumetric seamless map ϕ.

IV Quantize ϕ into an integer-grid map ϕ_q.

IVa Fix numerical errors, make ϕ *truly seamless*.

… Decompose the volume, ….

V Extract a hex mesh from the ϕ_q.



Seamless maps (or equivalently: seamless parametrizations) are commonly obtained through numerical optimization routines like eq. (3.5) [Nieser et al. 2011]. This involves inaccuracies due to limited precision and employed floating-point tolerances. Therefore, the resulting maps commonly are not exactly seamless on a numerical level, i.e., the continous versions of the constraints $\mathbf{IGM_{II}}$ to $\mathbf{IGM_{IV}}$ are only fulfilled up to some numerical tolerance ε. This bears potential of leading to inconsistencies for the volume decomposition algorithm introduced in the next chapter. In fact, the robustness of the decomposition algorithm can be ensured only when seamlessness constraints are *exactly* satisfied in the input. For the 2D case, this issue was discussed in detail in previous work [Ebke et al. 2013; Mandad and Campen 2019]. The latter article proposes a method that transforms a nearly seamless parametrization of a triangle mesh into one that is truly seamless—while altering it only locally and as little as possible. In this chapter we describe a generalization to the volumetric case on tetrahedral meshes.

4.1 Background: 2D Case

We briefly recapitulate the 2D case, focusing on the differences and referring to the original papers [Ebke et al. 2013; Mandad and Campen 2019] for a complete overview. The overall equality constraint system for 2D seamless parametrization consists of chart transition constraints across all non-boundary edges, alignment constraints along the boundary and feature edges, and possibly constraints of further type that have no correspondence with our setting. The relevant equality constraints and the involved variables form a highly underdetermined system

$$A\boldsymbol{x} = \mathbf{0} \qquad \text{(Seamlessness)}. \tag{4.1}$$

Solutions $\bar{\boldsymbol{x}}$ obtained in practice fulfill these constraints not exactly, but only up to some tolerance ε, hence $|A\bar{\boldsymbol{x}}| \leq \boldsymbol{\varepsilon}$. Typical values are $\varepsilon \leq 10^{-6}$. We say, the map is not *truly* seamless but just *almost* seamless. In terms of the objective function, solutions are also almost but not truly optimal.

The reason for this is that computers typically operate on floating point representations, which are inherently discrete, and true optimization on such a discrete domain would be NP-hard.

Still, the solution $\bar{x}$ can be expected to be close to other almost optimal but truly seamless solutions x^*. Hence, the idea is to not re-optimize the problem with exactness requirements, but rather perform a *sanitization*. This means finding a solution in the vicinity of the initial solution—and within the floating point realm—that is possibly slightly less objective-optimal, but exactly satisfies the constraints. In other words, the initial solution $\bar{x}$ is projected onto the intersection $\Omega \cap \mathbb{F}^n$ of the feasible set Ω of eq. (4.1) and the space of floating point solutions $\mathbb{F}^n$. Inequality constraints, like $\mathbf{IGM_I}$, are expected to already be fulfilled with enough margin so that their satisfaction is unaffected by the tiny corrections performed via projection.

Notably, the sanitization of 2D integer-grid maps [Ebke et al. 2013] (rather than more general *seamless* maps [Mandad and Campen 2019]) is a much easier special case; mainly, because knowing the intended solution is integer means rounding is a valid, fully local projection strategy. The projection for seamless maps instead requires globally consistent choices, as detailed below. For the same reason, simple 3D integer-grid map sanitization methods [Lyon et al. 2016] are not conducive to our setting; instead we aim to extend the approach of Mandad and Campen [2019] to 3D seamless maps.

Separating free and implied variables To obtain an exact solution to the constraint system in eq. (4.1), close to the given almost-seamless parametrization, [Mandad and Campen 2019] propose to first separate the variables into two sets—*implied* and *free*—by converting it into integer reduced row echelon form (IRREF). Concretely, this means $A \in \mathbb{Z}^{m \times n}$ with $m < n$ is transformed into an equivalent matrix $\hat{A} \in \mathbb{Z}^{m \times n}$ such that:

- Any row with a pivot (a first non-zero element) in column i, is either the first row, or the row above it has a pivot in column $j < i$.
- Each column containing a pivot contains no other non-zero entry.

Due to the A containing only integer entries, this can be done without numerical error, confined to the integer domain. In the resulting system free variables correspond to pivot-less columns. Assuming full row rank, the number of free variables equals $n - m$, i.e., results from the degree of underdeterminedness of the system. These free variables can be chosen freely, effectively eliminating the respective non-pivot columns. The remaining m implied variables are obtained as linear combinations of the chosen values of free variables, with rational coefficients of limited denominator magnitude.

Choosing free variables The crux is then to choose values for the free variables from $\mathbb{F}$ reasonably close to their values in the initial solution, but in a way which ensures that all additions, subtractions, multiplications and divisions within linear combinations can be carried out *exactly* in $\mathbb{F}$. Actually, addition, subtraction and multiplication can all be formulated as summations, while divisions need extra care. Any full solution, obtained by choosing free variables and computing implied variables via summation and division of free variables, then is truly seamless even in the floating-point domain.

The projection of the unsanitized initial values $\bar{x}_i$ of free variables onto safely summable and divisible floating point numbers $\mathbb{F}_{\text{safe}}$ is carried out as summarized in algorithm 4.1. In algorithm 4.1 PIVOTLCM$(i, \hat{A})$ is the least common multiple of pivot elements in rows of $\hat{A}$ that involve x_i. The global input K is the maximum floating point exponent that can occur at any point in the computation

during any deduction of implied variables from free variables. Assuming only small initial violations of constraints in the input—and by employing a smartly alternating sequence of algebraic operations [Mandad and Campen 2019, Alg.5]—K can be conservatively (over)estimated as $K = \max_i \lceil \log_2 |\bar{x}_i| \rceil$. Notably, algorithm 4.1 suffice for integer linear combinations; algorithm 4.1 are only required to make divisions safe when rational coefficients are involved.

Algorithm 4.1: SAFEPROJECTION

Input: free variable values $\boldsymbol{x} = (x_1, x_2, \dots)$, constraints $\hat{A}$ in IRREF, max exponent K
Output: $\boldsymbol{x}$ projected onto $\mathbb{F}_{\text{safe}}$, for which all variable values deduced via $\hat{A}$ remain in $\mathbb{F}$
foreach $x_i \in \boldsymbol{x}$ **do**
1 $\quad \ell_i \leftarrow$ PIVOTLCM$(i, \hat{A})$
2 $\quad x_i \leftarrow x_i / \ell_i$
3 $\quad x_i \leftarrow x_i + \text{sign}(x_i) 2^K$
4 $\quad x_i \leftarrow x_i - \text{sign}(x_i) 2^K$
5 $\quad x_i \leftarrow x_i \ell_i$

4.2 Extension to 3D

This approach is generic and in principle applicable to any homogeneous constraint system (including 3D seamlessness constraints). However, in naive formulation, seamlessness constraints form a large system with a size of the same order as the mesh; variables are the $\boldsymbol{u}$-parameters of the mesh's vertices, up to 2.5 million for extreme cases in the datasets used for this thesis. In such cases applying the above approach directly is impractical, because establishing IRREF for such a large system would be too time-consuming and values for ℓ_i could potentially grow quite large in the process, thus increasing the gap between initial and sanitized solution. [Mandad and Campen 2019] showed how a simplified core system (over only certain *sector* variables of particular *node* vertices) can instead be considered in the algorithm outlined above, drastically reducing the effective system size. The remaining variables are peripheral variables, and the remaining system of equations that involves them can be set up in a structure that allows direct sanitization without IRREF transformation or division.

Via generalization, we demonstrate that the same can be achieved in the 3D case. Concretely, the goal is to show that the constraint system can be set up as follows:

$$A\boldsymbol{x} = \left[\begin{array}{c|c} P & B \\ \hline \mathbf{0} & C \end{array}\right] \left[\begin{array}{c} \boldsymbol{x}_{\text{per}} \\ \hline \boldsymbol{x}_{\text{core}} \end{array}\right] = \mathbf{0}, \tag{4.2}$$

where C is a small core matrix, with a size depending only on the complexity of the singular graph and features in the input—rather than the size of the entire mesh. Meanwhile, P is very sparse and in integer row echelon form (not reduced, but that would be of no additional benefit). Hence, the peripheral variables $\boldsymbol{x}_{\text{per}}$ are separable a priori into free variables (pivotless columns of P) and implied variables (pivot columns of P). Crucially, the coefficients P_{ij}, B_{ij} are exclusively from $\{-1, 0, 1\}$, so all implied peripheral variables can be recovered as partial sums over previously chosen (signed) variables without involving divisions.

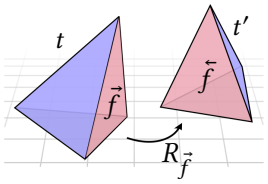
Note that this structure is achieved via careful system setup, not via expensive transformation operations. A system structured in this way enables an efficient algorithm, outlined as follows:

1.) Set up system $A\boldsymbol{x} = 0$ composed as above.
2.) Transform C into IRREF.
3.) Determine $K = \max_i \lceil \log_2 |\bar{x}_i| \rceil$ over all variables.
4.) Apply algorithm 4.1 of SAFEPROJECTION to find safe values for free core variables.
5.) Compute implied core variables via safe summation and division of free core variables.
6.) Apply only algorithm 4.1 of SAFEPROJECTION to find safe values for free peripheral variables.
7.) Compute implied peripheral variables via alternating sums of already chosen variables, starting from the lowest row of P and working upwards.

Below we detail how to setup the constraint system structured as in eq. (4.2).

4.2.1 Constraints

For our purpose, we are interested in the continous relaxations of equality constraints $\mathbf{IGM_{II}}$ to $\mathbf{IGM_{IV}}$ and the additional ones arising from alignment of feature facets $\hat{F} \in F$ and edges $\hat{E} \in E$, as discussed in the last paragraph of section 3.3. These constraints are homogenous and together can be expressed as $A\boldsymbol{x} = \mathbf{0}$, where $\boldsymbol{x}$ is a stack of per-tet vertex parameters $\boldsymbol{u}_{v,t}$. Inequality constraints $\mathbf{IGM_I}$ are expected to already be fulfilled with enough margin so that they will not be invalidated by the numerical sanitization.



Seamless transitions The seamless parametrization ϕ consists of affine maps $\phi_t : t \to \mathbb{R}^3$ per tet t, expressed via local per-vertex parameters $\boldsymbol{u}_{v,t}$. Across each oriented facet $\vec{f}$ connecting the tets $t, t' \in T$, parameters are related via transitions of the form:

$$\forall \vec{f} \in F : \forall v \in \partial^2 \vec{f} : \boldsymbol{u}_{v,t'} = R_{\vec{f}} \boldsymbol{u}_{v,t} + d_{\vec{f}}, \qquad R_{\vec{f}} \in \text{OCT}, d_{\vec{f}} \in \mathbb{R}^3, \tag{4.3}$$

i.e., continuous relaxations of eq. (3.3). The transition function across a facet in one direction $\vec{f}$ is the inverse of that across it in the opposite direction $\overleftarrow{f}$. Note that $R_{\vec{f}} \in \text{OCT}$ means that the rotation matrices $R \in \{-1, 0, 1\}^{3\times3}$ are simply (signed) permutations of coordinates, e.g., $R\boldsymbol{u} = R(u_1, u_2, u_3)^T = (-u_2, -u_3, u_1)$. The rotations of inverse transitions cancel out: $R_{\vec{f}} R_{\overleftarrow{f}} = R_{\overleftarrow{f}} R_{\vec{f}} = \mathbf{1}$, where $\mathbf{1}$ is the identity matrix.

Intended constraints The transition functions τ_f or alignment coordinates are typically not represented as explicit variables. Rather, they are implied by vertex parameters within charts. For arbitrarily non-seamless parametrizations one can not hope to deduce a well-fitting, intended transition function from inspecting the charts; no rotation might properly align oppposite charts. In our case, however, deviations from seamlessness are only present on a tiny numerical scale. Hence, as described for the 2D case [Ebke et al. 2013], the intended transition rotation stands out: Out of the 24 possible rotations from the octahedral group, there is only one that brings the edge vector mismatch to a value near zero. Hence, we can assume intended rotation to be given as:

$$R_{\vec{f}} = \operatorname*{arg\,min}_{R \in \text{OCT}} \sum_{v_1, v_2 \in \partial^2 \vec{f}} \left\| \boldsymbol{u}^{t'}_{v_2} - \boldsymbol{u}^{t'}_{v_1} - R_{\vec{f}}(\boldsymbol{u}^{t}_{v_2} - \boldsymbol{u}^{t}_{v_1}) \right\|. \tag{4.4}$$

The intended alignment coordinates of feature facets, feature edges and singular edges are determined in a similar manner.

4.2.1.1 Constraint Types

Transition seamlessness constraints The translations variables $d_{\vec{f}}$ in eq. (4.3) can be eliminated, by demanding that rotations $R_{\vec{f}}$ apply to edge vectors rather than vertex coordinates:

$$\boldsymbol{u}^{t'}_{v_2} - \boldsymbol{u}^{t'}_{v_1} = R_{\vec{f}}(\boldsymbol{u}^{t}_{v_2} - \boldsymbol{u}^{t}_{v_1}). \tag{4.5}$$

This adds three constraint equations per interior facet f.

Surface alignment constraints Both boundaries and feature surfaces are required to lie on parametric isoplanes, i.e., the map per tet t is required to map vertices v_1, v_2, v_3 of boundary and feature facets $f \in \partial\mathcal{M} \cup \hat{F}$ onto the same value in a given coordinate k:

$$\mathbf{1}_k(\boldsymbol{u}^{t}_{v_2} - \boldsymbol{u}^{t}_{v_1}) = \mathbf{1}_k(\boldsymbol{u}^{t}_{v_3} - \boldsymbol{u}^{t}_{v_1}) = 0, \tag{4.6}$$

where $\mathbf{1}_k \in \{0,1\}^3$ is a unit row vector extracting the k-th coordinate of $\boldsymbol{u}$. Accounting for redundancy, this amounts to one constraint equation per edge on the boundary or on a feature facet.

Curve alignment constraints Both singularities and feature curves of index 4, 8, ...are required to lie on parametric isolines, i.e., the map per tet t is required to map vertices v_1, v_2 of singular and feature edges onto the same value in two given coordinates i and j:

$$\mathbf{1}^i(\boldsymbol{u}^t_{v_2} - \boldsymbol{u}^t_{v_1}) = \mathbf{1}^j(\boldsymbol{u}^t_{v_2} - \boldsymbol{u}^t_{v_1}) = 0. \tag{4.7}$$

Accounting for redundancy, this yields two constraint equations per feature edge or singular edge with index $4n$.

4.2.2 Core-Periphery Identification

As discussed before, the naive constraint formulation yields a system that is unnecessarily large and complex. Both the number of constraints and variables can be trimmed, the remaining variables can be separated into *core* and *peripheral* variables and the equation system can be set up in neat form. The goal is hence to identify or construct *core* and *periphery*.

The cut-align complex As introduced in section 3.3, we call any facet for which the intended transition function is not the identity a *cut facet*, and the union of all such facets F_{cut} the *cut surface*. Facets on the mesh boundary $\partial\mathcal{M}$ and feature facets are associated with alignment constraints, hence we call them *align facets* F_{align}. Feature edges and singularities with index $4n$ similarly are called *align edges* E_{align}. Consider the restriction $\mathcal{M}|_{F_{\text{cut}} \cup F_{\text{align}} \cup E_{\text{align}}}$, which we call the *cut-align complex*. This cell complex is of dimension 2 but is generally not pure and not necessarily connected; chains of feature edges may occur which are not incident to a facet in the restriction and features may occur isolated from the cut surface. Singularities are, by definition, included in the complex and are connected via the cut surface to the mesh boundary and vice-versa.

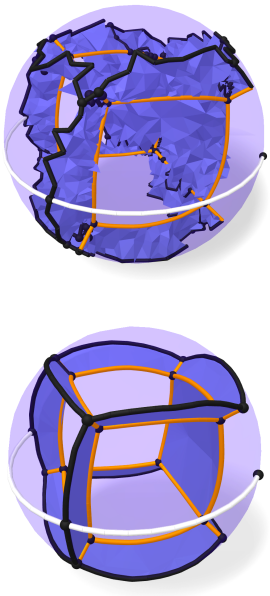
Node-branch-sheet structure We call an edge of the cut-align complex a *branch edge* if one of the following holds:

- it is a singular edge,
- it is an align edge,
- it is an edge incident to (at least) two differently aligned align facets,
- it is a boundary or non-manifold edge within the restriction.

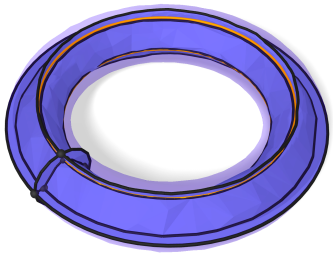
We furthermore refer to a vertex of the cut-align complex as a *node* if any of the following holds:

- it is incident to just one or to more than two branch edges,
- it is incident to two differently aligned feature edges,
- it is incident to just one feature or singular edge,
- it is a non-manifold vertex in the restriction.

A connected set of branch edges bounded by nodes form a *branch*. All these branches together partition the facets of the cut-align complex into maximum pieces we call *sheets*.

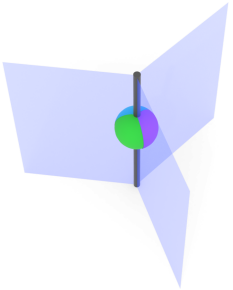


By construction, each sheet is an orientable 2-manifold surface (completely bounded by branches) and is either a pure *cut sheet*, a pure *align sheet*, or a *mixed sheet* that is both, cut and aligned, everywhere. Also, within each sheet the transition function and (potentially) the aligned coordinate are constant. Furthermore, each branch is a manifold curve (bounded by nodes) and anywhere along its length is incident to the same cycle of sheets. If the branch is incident to no or only to align sheets, we call it an *align branch*, if it is incident to only cut sheets, we call it a *cut branch*, otherwise it is a *mixed branch*. The inset demonstrates the layout of sheets, branches and nodes for an example input. The original input is shown at the top; for visual clarity, the bottom image shows a layout obtained from a cut surface edited for minimal area. Align sheets only exist on the boundary in the given example, the remaining blue sheets are cut sheets. Singular (cut) branches are highlighted in orange, feature (align) branches in white. Black branches are mixed, as they are incident to both boundary (align) and cut sheets.



Node presence For align branches, the nature of seamless and integer grid maps ensures the presence of nodes. Node-less align loops can not occur, because that would require a cycle with alignment to the same coordinate, which implies an intersection with the cut surface, i.e., a node (see the inset). Closed align sheets without nodes are precluded with a similar argument: The surface would need to be aligned with the same coordinate globally, which for topological spheres according to the Gauss-Bonnet theorem (equivalently, the Poincaré-Hopf theorem of the respective cross field [Ray et al. 2008]) requires a singularity to pierce (or end at) the surface, implying a node, and for more complex topologies implies cutting along (at least) two independent cycles to yield a disk, again implying a node at the cycle intersection.

This leaves only node-less loop branches resulting from two sheets intersecting and node-less, closed cut sheets. While not very frequent, these can indeed occur, so an additional vertex is marked as a node anywhere on these to ensure they are included in the following node-based considerations.



Sectors Around each vertex of the (non-restricted) tet mesh, the cut facets partition the tetrahedral mesh into *sectors*, such that all incident tetrahedra within a sector (should) share parametrization values at the vertex. When that is not the case, i.e., parameters of the same vertex are not already equal across non-cut facets, they can trivially be matched by setting parameters on either side of such facets to those on the opposite side and consistently spreading the assignment via non-cut facets incident to the vertex. Hence, a single parameter representation per sector is sufficient also for the simplified constraint system we aim to build; such a sector-based representation is often used for seamless parametrizations [Bommes et al. 2009]. We call the parameters in sectors of nodes *core parameters*, and those in sectors of non-node branch or sheet vertices *peripheral parameters*. Notably, the parameters of other vertices, i.e., those not part of the cut-align complex, are completely decoupled from other parameters, and can thus trivially be sanitized as described above; the constraint system considered from here on includes only sector parameters of vertices in the cut-align

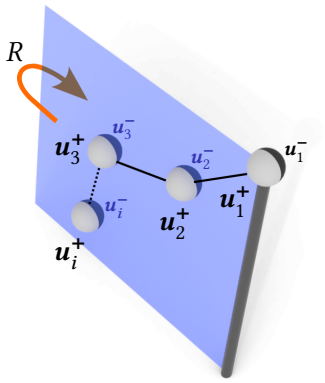
complex. This already implies a major reduction in system size with respect to the naive full system.

4.2.3 Core-Periphery System Setup

Generalizing [Mandad and Campen 2019, §5.3] to 3D, we demonstrate how to set up the system as outlined in eq. (4.2), where P is in integer row echelon form and coefficients P_{ij}, B_{ij} are exclusively from $\{-1, 0, 1\}$. Then the full-blown sanitization only needs to be applied to the small subsystem $C\boldsymbol{x}_{\text{core}}$. Afterwards, peripheral variables $\boldsymbol{x}_{\text{per}}$ are either chosen via projection or efficiently deduced by traversing P from bottom to top.

4.2.3.1 Cut Sheets

For each sheet we can denote the parameters of a vertex v_i in the two sectors on the two sides of the sheet as $\boldsymbol{u}_i^+$ and $\boldsymbol{u}_i^-$. Given a sheet with transition rotation R, the transition constraint for an edge between v_i and v_j on the sheet then has the form $\boldsymbol{u}_j^- - \boldsymbol{u}_i^- = R(\boldsymbol{u}_j^+ - \boldsymbol{u}_i^+)$. Given a sequence of vertex parameters $(\boldsymbol{u}_1^\pm, \boldsymbol{u}_2^\pm, \ldots, \boldsymbol{u}_n^\pm)$ forming a chain of edges on a sheet, the equation corresponding to the i-th edge is: $\boldsymbol{u}_i^- - \boldsymbol{u}_{i-1}^- = R(\boldsymbol{u}_i^+ - \boldsymbol{u}_{i-1}^+)$. Cumulative sums of these equations along the sequence have a simple form, namely



$$\boldsymbol{u}_i^- - \boldsymbol{u}_1^- = R(\boldsymbol{u}_i^+ - \boldsymbol{u}_1^+). \tag{4.8}$$

All transition constraints of a sheet can hence be rewritten relative to one selected *base* core node of the sheet, with parameters $\boldsymbol{u}_1^\pm$. Re-ordering the variables in the system such that core parameters come last, the following structure is obtained per sheet:

$$\left[\begin{array}{c|c} P_1 & B_1 \\ \hline \mathbf{0} & C_1 \end{array}\right] = \left[\begin{array}{cccc|cccc} \multicolumn{4}{c|}{\boldsymbol{u}_n^- \ldots \boldsymbol{u}_{m+1}^-} & \multicolumn{4}{c}{\boldsymbol{u}_m^- \ldots \boldsymbol{u}_2^- \quad \boldsymbol{u}_1^-} \\ \mathbf{1} & & -R & & & & -\mathbf{1} & \\ & \ddots & & \ddots & & & \vdots & \\ & \mathbf{1} & & -R & & & -\mathbf{1} & \\ \hline & & & & \multicolumn{4}{c}{\boldsymbol{u}_m^+ \ldots \boldsymbol{u}_2^+ \quad \boldsymbol{u}_1^+} \\ & & & & \mathbf{1} & & -\mathbf{1} & -R \\ & & & & & \ddots & \vdots & \\ & & & & & \mathbf{1} & -\mathbf{1} & \end{array}\right] \begin{array}{cccc|cccc} \multicolumn{4}{c|}{\boldsymbol{u}_n^+ \ldots \boldsymbol{u}_{m+1}^+} & \multicolumn{4}{c}{\boldsymbol{u}_1^+} \\ & & & & & & & R \\ & & & & & & & \vdots \\ & & & & & & & R \\ \hline & & & & & & & R \\ & & & & & & & \vdots \\ & & & & -R & & & R \end{array} \tag{4.9}$$

where each entry is a 3×3 block ($R \in \{-1, 0, 1\}^{3\times3}$: signed permutation matrix; $\mathbf{1} = \text{diag}(1, 1, 1)$), because $\boldsymbol{u}$ has three components. The vertical bar separates core and peripheral variables, the horizontal one separates constraints that only involve core variables from the rest. The system matrix has $3n$ rows and $6n$ columns, where n is the number of vertices in the sheet, with inner vertices of branches excluded (these are covered further below). The number of nodes among these vertices is m, and the submatrix C_1 has $3m$ rows and $6m$ columns.

4.2.3.2 Align Sheets

Given a sheet with aligned coordinate k, the alignment constraint for an edge between v_i and v_j on the sheet then has the form $\mathbf{1}_k(\boldsymbol{u}_j - \boldsymbol{u}_i) = 0$. Analogously, cumulative sums of alignment constraints relative to a base node can be built, forming this structure per align sheet after sorting

peripheral and core variables:

$$\left[\begin{array}{c|c} P_2 & B_2 \\ \hline \mathbf{0} & C_2 \end{array}\right] = \begin{array}{c} \begin{array}{ccc} \boldsymbol{u}_n \ldots \boldsymbol{u}_{m+1} & \boldsymbol{u}_m \ldots \boldsymbol{u}_2 & \boldsymbol{u}_1 \end{array} \\ \left[\begin{array}{ccc|ccc} \mathbf{1}_k & & & & & -\mathbf{1}_k \\ & \ddots & & & & \vdots \\ & & \mathbf{1}_k & & & -\mathbf{1}_k \\ \hline & & & \mathbf{1}_k & & -\mathbf{1}_k \\ & & & & \ddots & \vdots \\ & & & & \mathbf{1}_k & -\mathbf{1}_k \end{array}\right] \end{array} \tag{4.10}$$

where $\mathbf{1}_k$ is a 1×3 block: $[1, 0, 0]$ for $k = 1$, $[0, 1, 0]$ for $k = 2$, $[0, 0, 1]$ for $k = 3$. The entire matrix has n rows and $3n$ columns, C_2 accounting for m rows and $3m$ columns, where n and m are again the number of non-inner-branch vertices and node vertices, respectively. Here, it is possible that the sheet is self-adjacent at some nodes, at which a cut sheet must create two sectors, in which case n and m more precisely refer to the number of vertex sectors on the sheet side.

4.2.3.3 Mixed Sheets

Given a sheet with aligned coordinate k and transition rotation R, combination of the above yields:

$$\left[\begin{array}{c|c} P_3 & B_3 \\ \hline \mathbf{0} & C_3 \end{array}\right] = \begin{array}{c} \begin{array}{cccccccc} \boldsymbol{u}_n^- \ldots \boldsymbol{u}_{m+1}^- & \boldsymbol{u}_n^+ \ldots \boldsymbol{u}_{m+1}^+ & \boldsymbol{u}_m^- \ldots \boldsymbol{u}_2^- & \boldsymbol{u}_1^- & \boldsymbol{u}_m^+ \ldots \boldsymbol{u}_2^+ & \boldsymbol{u}_1^+ \end{array} \\ \left[\begin{array}{cc|cccc} \mathbf{1} & -R & & -\mathbf{1} & & R \\ & \ddots & & \vdots & & \vdots \\ & \mathbf{1} \quad -R & & -\mathbf{1} & & R \\ & \mathbf{1}_k & & & & -\mathbf{1}_k \\ & \ddots & & & & \vdots \\ & \mathbf{1}_k & & & & -\mathbf{1}_k \\ \hline & & \mathbf{1} & -\mathbf{1} & -R & R \\ & & \ddots & \vdots & \ddots & \vdots \\ & & \mathbf{1} & -\mathbf{1} & -R & R \\ & & & & \mathbf{1}_k & -\mathbf{1}_k \\ & & & & \ddots & \vdots \\ & & & & \mathbf{1}_k & -\mathbf{1}_k \end{array}\right] \end{array} \tag{4.11}$$

Note that only aligning the positive side of the sheet is sufficient as this implies alignment of the negative side as well. This system has $4n$ rows and $6n$ columns, to which C_3 contributes $4m$ rows and $6m$ columns.

4.2.3.4 Pure Align Branches

Along a branch with aligned coordinates k and l, irrespective of whether this is already covered by incident align sheets or (partially) achieved by additional constraints, the constraint system is

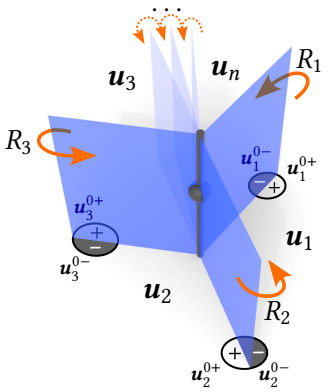
composed from two versions of eq. (4.10):

$$\left[\begin{array}{c|c} P_4 & B_4 \\ \hline \mathbf{0} & C_4 \end{array}\right] = \begin{array}{c} \begin{matrix} \boldsymbol{u}_n & \dots & \boldsymbol{u}_3 & \boldsymbol{u}_2 & \boldsymbol{u}_1 & \boldsymbol{u}_0 & \boldsymbol{u}_0' \end{matrix} \\ \left[\begin{array}{ccccccc} \mathbf{1}_k & & & & & & -\mathbf{1}_k \\ \mathbf{1}_l & \ddots & & & & -\mathbf{1}_l & \vdots \\ & & \mathbf{1}_k & & & & -\mathbf{1}_k \\ & & \mathbf{1}_l & & & -\mathbf{1}_l & \\ \hline & & & \mathbf{1}_k & & & -\mathbf{1}_k \\ & & & \mathbf{1}_l & & -\mathbf{1}_l & \\ & & & & \mathbf{1}_k & & -\mathbf{1}_k \\ & & & & \mathbf{1}_l & -\mathbf{1}_l & \end{array}\right] \end{array} \tag{4.12}$$

This matrix is in row echelon form when choosing $k < l$. Here, n is the number of vertex sectors along the branch, two of those being node sectors. In case of a cyclic branch, the first and last node are the same, but must lie at a cut; hence, the sector parameters are still different ones. The system has between $2n-2$ and $2n$ rows and between $3n$ and $3n+6$ columns. This is because the base parameters $\boldsymbol{u}_0$ and $\boldsymbol{u}_0'$, and one branch node parameter $\boldsymbol{u}_1$, could be pairwise different or refer to the same, in which case the columns would be merged via summation and resulting 0-rows could be dropped. If the align branch is isolated, i.e., not formed by incident align sheets, $\boldsymbol{u}_0$, $\boldsymbol{u}_0'$ and $\boldsymbol{u}_1$ are the same, so only six columns remain to the right of the vertical bar. Then, the two rows and six columns of C_4 are not covered by sheets yet and need to also be added to the global system.

4.2.3.5 Cut Branches

We consider a configuration of branches as shown in the inset: n sector variables $\boldsymbol{u}_1 \dots \boldsymbol{u}_n$ are ordered in a cycle, and separated by n cut sheets that have transition rotations $R_1 \dots R_n$ and base nodes $v_1^0 \dots v_n^0$, each with positive and negative sector base parameters $\boldsymbol{u}_1^{0\pm} \dots \boldsymbol{u}_n^{0\pm}$. Any two sectors i and $i+1$ are neighboring across sheet $i+1$. Sector indices are understood as modulo n, to ensure cyclicity. Then constraints on $\boldsymbol{u}_i$ are formulated relative to the negative base parameter $\boldsymbol{u}_{i+1}^{0-}$, and on $\boldsymbol{u}_{i+1}$ relative to the positive base parameter $\boldsymbol{u}_{i+1}^{0+}$. Concretely, the governing equation across sheet $i+1$ is



$$\boldsymbol{u}_i - \boldsymbol{u}_{i+1}^{0-} = R_{i+1}(\boldsymbol{u}_{i+1} - \boldsymbol{u}_{i+1}^{0+}). \tag{4.13}$$

Rebasing Each sheet having its own base node prevents us from straightforwardly relating variables across multiple sheets, e.g, $\boldsymbol{u}_i$ to $\boldsymbol{u}_{i+2}$. This would be simple if all sectors shared the same base node. In fact, we can simply choose one of the branch nodes as a new base node for the constraint system on branch nodes. In the following, this rebasing process is demonstrated algebraically. The new base node, as it also lies on the branch and hence on all sheets incident to it, exhibits the same sectors as in the inset. Hence, it has sector parameters, which are denoted $\boldsymbol{u}_{1\dots n}^0$, and no longer require positive/negative sheet side specification. Because the new base node is contained in all incident sheets, there exists a sheet-internal path from any previous base node to it, characterized by the cumulative sum

$$\boldsymbol{u}_{i+1}^{0-} - \boldsymbol{u}_i^0 = R_{i+1}(\boldsymbol{u}_{i+1}^{0+} + \boldsymbol{u}_{i+1}^0). \tag{4.14}$$

Summation of eqs. (4.13) and (4.14) gives

$$\boldsymbol{u}_i - \boldsymbol{u}_i^0 = R_{i+1}(\boldsymbol{u}_{i+1} - \boldsymbol{u}_{i+1}^0). \tag{4.15}$$

which now permits straightforward chaining of equations, e.g., $\boldsymbol{u}_1 - \boldsymbol{u}_1^0 = R_2(\boldsymbol{u}_2 - \boldsymbol{u}_2^0) = R_2R_3(\boldsymbol{u}_3 - \boldsymbol{u}_3^0)$.

Equation system setup Given the rebased formulation and using the shorthand $\Delta\boldsymbol{u}_i = \boldsymbol{u}_i - \boldsymbol{u}_i^0$, the set of equations governing the sector parameters along pure cut branches are

$$\begin{aligned} \epsilon_1 : \quad \Delta\boldsymbol{u}_1 &= R_2\Delta\boldsymbol{u}_2 \\ \epsilon_2 : \quad \Delta\boldsymbol{u}_2 &= R_3\Delta\boldsymbol{u}_3 \\ &\dots \\ \epsilon_{n-1} : \quad \Delta\boldsymbol{u}_{n-1} &= R_n\Delta\boldsymbol{u}_n \\ \epsilon_n : \quad \Delta\boldsymbol{u}_n &= R_1\Delta\boldsymbol{u}_1 \end{aligned} \tag{4.16}$$

This is a cyclic dependency of variables, which can be unrolled by substituting all prior equations into the last one, yielding

$$\epsilon_n : \ \Delta\boldsymbol{u}_n = R_1R_2 \dots R_n\Delta\boldsymbol{u}_n$$

as the new final equation. $R_{1\dots n} \coloneqq R_1R_2 \dots R_n$ is the product of (signed) permutation matrices, hence is itself a (signed) permutation matrix. A case analysis reveals:

If $R_{1\dots n} \neq \mathbf{1}$ the branch must be made of singular edges (with non-divisible by 4 index). In this case seamless map topology implies that $R_{1\dots n}$ must be a rotation around one of the principal axes [Nieser et al. 2011], i.e., $R_{1\dots n} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}^n$ for $n \in \{1, 2, 3\}$ (up to simultaneous row-and-column swaps). Then, $\mathbf{1} - R_{1\dots n}$ is either $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ or $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$, for $n = 1 \dots 3$ respectively. In either case, the final equation $\epsilon_n : \ \Delta\boldsymbol{u}_n(\mathbf{1} - R_{1\dots n}) = 0$ is equivalent to $\mathbf{1}_k\Delta\boldsymbol{u}_n = \mathbf{1}_l\Delta\boldsymbol{u}_n = 0$, i.e., alignment constraints on two coordinates of $\boldsymbol{u}_n$. This is confirmed by prior knowledge: Singularities of non-divisible-by-4 index are (inherently) aligned with parametric isolines. In case of mixed branches, additional alignment sheets incident on the branch could add only redundant constraints and need no special case handling.

If $R_{1\dots n} = \mathbf{1}$ the branch is made of regular edges or of multiple-of-4 singularities; The equation $\epsilon_n : \Delta\boldsymbol{u}_n(\mathbf{1} - R_{1\dots n}) = 0$ is trivial and can be dropped. Assuming mixed branches which are also incident to alignment sheets, up to two alignment constraints (more would be redundant) could exist for sector variables i and j: $\mathbf{1}_k\Delta u_i = \mathbf{1}_l\Delta u_j = 0$ Both can be transformed into constraints on the last sector $\mathbf{1}^{k'}\Delta u_n = \mathbf{1}^{l'}\Delta u_n = 0$ via $\mathbf{1}^{k'} = \pm\mathbf{1}_kR_{i\dots n}$ and $\mathbf{1}^{k'} = \pm\mathbf{1}_kR_{j\dots n}$ (omitting the potentially negative sign is without consequence). This is, in fact, equivalent to the situation where the cycle is not closed at all, i.e., where the cut sheet with $i = 1$ that would connect the sectors $i = n$ and $i = 1$, is replaced by two boundary sheets, separating the sectors. These boundary sheets then carry two alignment constraints $\mathbf{1}_k\Delta u_1 = \mathbf{1}_l\Delta u_n = 0$, and the former can be transformed into $\mathbf{1}^{k'}\Delta u_n = 0$ via $\mathbf{1}^{k'} = \mathbf{1}_kR_{2\dots n}$.

Subsystem structure The above case analysis shows that for both pure and mixed cut branches, as well as both boundary and interior cut branches the equation system is at most

$$\left[\begin{array}{c|c} P_5 & B_5 \end{array}\right] = \begin{array}{c} \begin{matrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \dots & \boldsymbol{u}_{n-1} & \boldsymbol{u}_n \end{matrix} \\ \left[\begin{array}{ccccc|ccccc} \mathbf{1} & -R_2 & & & & -\mathbf{1} & R_2 & & & \\ & \ddots & \ddots & & & & \ddots & \ddots & & \\ & & \ddots & \ddots & & & & \ddots & \ddots & \\ & & & \mathbf{1} & -R_n & & & & -\mathbf{1} & R_n \\ & & & & \mathbf{1}_k & & & & & -\mathbf{1}_k \\ & & & & \mathbf{1}_l & & & & & -\mathbf{1}_l \end{array}\right] \end{array} \begin{array}{c} \begin{matrix} \boldsymbol{u}_1^0 & \boldsymbol{u}_2^0 & \dots & \boldsymbol{u}_n^0 & \boldsymbol{u}_n^0 \end{matrix} \\ \\ \end{array} \tag{4.17}$$

with the last two rows occurring for singular branches, and one or both of the last rows occurring for mixed (inner or boundary) branches.

4.2.3.6 Global System

Combining the systems over all branches and sheets yields the global system. When non-node branch variables and constraints involving them are omitted from the sheet systems (they are already covered in the branch systems) no peripheral variable appears in more than one system. Hence, when combining all previous systems and ordering columns so that the peripheral matrices P_i are contiguous, and columns pertaining to B_i and C_i come last, the global system has the following structure:

$$\left[\begin{array}{c|c} P & B \\ \hline \mathbf{0} & C \end{array}\right] = \left[\begin{array}{ccccc|c} P_1 & & & & & B_1 \\ & P_2 & & & & B_2 \\ & & P_3 & & & B_3 \\ & & & P_4 & & B_4 \\ & & & & P_5 & B_5 \\ \hline & & & & & C_1 \\ & & & & & C_2 \\ & & & & & C_3 \\ & & & & & C_4 \end{array}\right] \begin{array}{l} \text{Cut sheets: eq. (4.9)} \\ \text{Align sheets: eq. (4.10)} \\ \text{Mixed sheets: eq. (4.11)} \\ \text{Align branches: eq. (4.12)} \\ \text{Cut branches: eq. (4.17)} \\ \\ \text{Cut sheets: eq. (4.9)} \\ \text{Align sheets: eq. (4.10)} \\ \text{Mixed sheets: eq. (4.11)} \\ \text{Isolated align branches: eq. (4.12)} \end{array}$$

Here, P is in row echelon form and composed of coefficients from $\{-1, 0, 1\}$ because each $P_{1\dots5}$ is; the same is true for coefficients of B.

4.3 Implementation Notes

When implementing the setup of the above system, as well as its exact solution via projection, it is convenient to only explicitly setup and solve the system $C\boldsymbol{x}_{\text{core}}$. The setup of P and B, as well as algebraic operations necessary to then obtain $\boldsymbol{x}_{\text{per}}$ from $\boldsymbol{x}_{\text{core}}$ can be combined (implicitly) by propagating the solution of node sector variables into sheet and branch sector variables directly via the mesh connectivity. This avoids overhead from explicitly setting up P and B. Concretely, given a numerically exact solution for node sector variables $\boldsymbol{x}_{\text{core}}$, a connectivity-based solution propagation can be executed as follows

1.) For every align sheet, set the aligned coordinate of every non-node sector on the sheet to that of any node sector on the same sheet side.

2.) For every align or singular branch, set both aligned coordinates of every branch-interior vertex sector to those of any equivalent node sector on the branch.

3.) For every cut sheet or mixed sheet:

   a) Determine the translational components d of the sheet's transition function as

$$d_f = \boldsymbol{u}_0^- - R_f \boldsymbol{u}_0^+,$$

   where $\boldsymbol{u}_0^\pm$ are the two sector parameters of any node on the sheet. Since R_f is a signed permutation, no division is involved.

   b) For every non-node, non-branch vertex on the sheet with sector parameters $\boldsymbol{u}_i^\pm$:

      i. For the positive sector variables $\boldsymbol{u}^+$, use lines algorithm 4.1 of algorithm 4.1 to choose floating point values, safe under summation and subtraction, skipping aligned coordinates.

      ii. For the negative sector variables set $\boldsymbol{u}^- \leftarrow R_f \boldsymbol{u}^+ + d_f$, skipping aligned coordinates.

4.) For every cut branch or mixed branch:

   a) For each non-node vertex on the branch:

      i. Order the sector n variables of that vertex according to the cycle of $(n-1)$ incident cut sheets.

      ii. For the first sector variables $\boldsymbol{u}_1$, use lines algorithm 4.1 of algorithm 4.1 to choose floating point values, safe under summation and subtraction, skipping aligned coordinates.

      iii. For any subsequent sector variables $\boldsymbol{u}_i$, separated by sheet i from the previous sector, set $\boldsymbol{u}_i \leftarrow R_{i-1}\boldsymbol{u}_{i-1} + d_{i-1}$, skipping aligned coordinates.

This assigns values to all non-node sector variables, i.e., exactly to $\boldsymbol{x}_{\text{per}}$, while ensuring that all constraints involving these variables are exactly satisfied. A reference implementation of the entire volumetric sanitization procedure has been made available as the open source library TRULYSEAMLESS3D[1].

[1] https://github.com/HendrikBrueckler/TrulySeamless3D

Map-guided Cuboid Decomposition

5

III Compute a volumetric seamless map ϕ

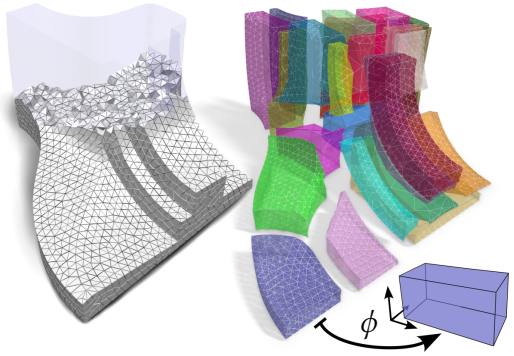
IV Quantize ϕ into an integer-grid map ϕ_q.

IVa Make ϕ *truly seamless*.

IVb Partition the volume into ϕ-aligned cuboids

… Obtain ϕ_q by quantizing the partition …

V Extract a hex mesh from ϕ_q.

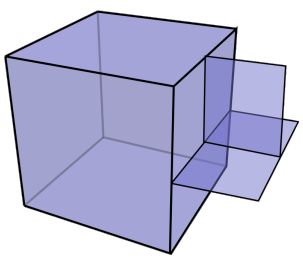


As demonstrated in the last chapter decomposing a problem and the domain it is defined on into smaller parts can make the difference between tractibility and intractability. In the context of quantization, i.e., transformation of a seamless map into a reasonably similar integer-grid map, a coarse domain decomposition is useful from several standpoints:

1. It allows cheap exploration of partial solutions to the integer-grid map problem. The mapping problem itself has a large number of continuous and relatively small number of integer variables. Working with an adequately coarse proxy, on which specifically integer assignments can be explored, allows cheap iterative refinement of solutions without the need for computing an entire map in each iteration.

2. By explicitly representing singularities and features as partition elements it provides compact connectivity information between such points, usable for efficient relation queries or constraint validation.

3. It allows the final integer-grid map to be computed in a component-based manner, computing a topologically trivial map per block of the decomposition and then stitching these maps back together into arbitrarily complex topologies.



This chapter, based on *truly seamless* maps as an input, establishes a robust construction routine for coarse block decompositions of the input domain, similar in concept to the base complex (see section 2.3.3.1). Each block of the partition is a cuboid in parameter space—each block side lies on a parametric iso-plane—but, different from the base complex, the blocks fill the domain in a non-conforming manner, which implies T-shaped configurations of block sides (rather than +-shaped ones).

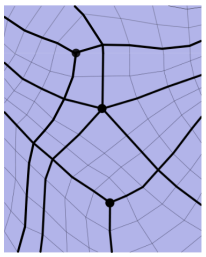
5.1 Surface Case

The idea of decomposing a domain with the help of a directional guiding structure defined on it, has been explored in various ways also in the surface setting. Most relevant to our case are approaches that construct a partition into quadrilateral patches. Such approaches require a guiding structure that defines directions of four-fold rotation symmetry (4-RoSy), most relevantly:

- cross and frame fields,
- seamless and integer-grid parametrizations, or
- quadrilateral meshes.

All the above define two pairs of opposite directions everywhere, except at a number of isolated singularities (where multiple outgoing directions are still well-defined, but opposite-relations are not). The idea is, in general, to expand region-separating curves that start from singularities along each outgoing direction, and are propagated by following at any given point the opposite of the direction along which the line arrived. Given a feasible stopping criterion for curve expansion, the final curve patterns partition the surface into patches that all are *four-sided*, completely *regular* in their interior, and *aligned* with the field's streamlines, the parametrization's iso-lines, or the mesh's edges, respectively [Eppstein et al. 2008]. Moreover, singularities and boundaries are explicitly represented as discrete elements of the high-level partitions. Additional curves can be expanded from feature points or placed along feature lines to ensure they are preserved, too.

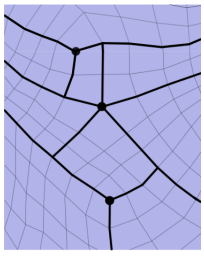
5.1.1 Base Complex



The base complex [Bommes et al. 2011, §2.2] of a surface with 4-RoSy guiding structure is known to be the coarsest *conforming* partition into four-sided, regular, aligned patches. It is obtained by stopping the expansion of a curve only once it hits a boundary or a singularity. This is well-defined without adjustments for quantized structures like integer-grid maps and quad meshes. For these, the fact that singularities lie on one of finitely many discrete alignment levels ensures that curves starting from singularities do not expand infinitely but at some point are bound collide with a singularity.

Still, spiral patterns of traces can often be observed [Bommes et al. 2011], in which a curve cycles around a surface handle multiple times before looping back to its initial singularity, yielding an overly fine decomposition in many cases. For non-quantized structures, like cross fields and seamless parametrizations, it can take arbitrarily long for such spiral traces to close the loop (when using limited precision, otherwise even infinite spirals are possible). In such cases, it is common to snap spirals to singularities, once they pass sufficiently close to one. Even so, the base complex can become impractically dense for applications running on limited memory, let alone for ones requiring a coarse scaffold of the input surface.

5.1.2 Motorcycle Graph



A *canonical* non-conforming partition of surfaces, the so called *motorcycle graph* [Eppstein and Erickson 1999; Eppstein et al. 2008], has been used in various computer graphics and geometry processing applications. In the canonical construction routine, the separating curves are the traces left behind by particles (called *motorcycles*) started at singularities. Here, particles are also stopped when they collide with a trace, thereby locally forming T-joints, which explains the non-conforming nature of the partition. Therefore the motorcycle graph is clearly a subgraph of the graph described by the base complex

on the same surface. Canonicity is achieved by demanding that particles start simultaneously, travel at equal speed and resolve exact orthogonal collisions via clockwise priority.

For use cases that are able to handle non-conformity (or even benefit from it), the motorcycle graph provides a major advantage over the base complex: It is often much simpler (having a smaller number of patches), in some cases even by orders of magnitude. Furthermore, the number of patches is within a constant factor of the minimum number possible for any partition with these properties, while finding a truly minimal partition is known to be much harder than computing the motorcycle graph [Eppstein et al. 2008].

5.2 Related Work: Structured Decomposition

The original 2D Euclidean definition of the motorcycle graph goes back to work by Eppstein and Erickson [1999]. It was extended to curved surfaces, i.e., Riemannian manifolds, initially for the purpose of quadrilateral mesh partitioning [Eppstein et al. 2008]. In this setting the mesh's edges provide the directional information guiding the motorcycles across the surfaces. This quad mesh driven motorcycle graph has been employed in further contexts, for reverse engineering [Gunpinar et al. 2014], texture mapping [Schertler et al. 2018], computational fabrication [Liu et al. 2020], and quadrilateral remeshing [Razafindrazaka and Polthier 2017; Nuvoli et al. 2019].

The idea of the motorcycle graph has also been adapted to surfaces equipped with other directional guiding structures. In particular, motorcycles following streamlines of a cross field [Vaxman et al. 2016] have been used for the reliable generation of global seamless surface parametrizations [Myles et al. 2014]. Motorcycles following the iso-lines of such seamless parametrizations [Kälberer et al. 2007; Bommes et al. 2009; Myles and Zorin 2012], in turn, have been used for the purpose of robust quantization [Campen et al. 2015; Lyon et al. 2019, 2021a,b]. A generalized class of parametrizations, so-called seamless similarity parametrizations, provide another structure that can be used to guide motorcycles [Campen and Zorin 2017]. This has been leveraged to construct surface T-meshes that can serve as domain for the definition of T-spline spaces [Campen and Zorin 2017; Karčiauskas et al. 2017].

For the 3D volumetric case, a concept analogous to the 2D motorcycle graph has not been described yet. Generalizations of the above mentioned guiding structures, however, exist. Hexahedral meshes can be considered the natural generalization of quadrilateral meshes to the next dimension. Cross fields generalize to octahedral (or more general 3D frame) fields [Huang et al. 2011; Solomon et al. 2017; Liu et al. 2018; Corman and Crane 2019; Zhang et al. 2020], and seamless parametrizations of triangular surface meshes extend naturally to tetrahedral volume meshes as well [Nieser et al. 2011], see also sections 3.5 and 3.6. So far only the base complex [Bommes et al. 2011] has been considered in a 3D setting, for the case of hex meshes [Gao et al. 2015a]. For hex meshes with many details, this structure can be highly complex; even more so for seamless volume parametrizations, where it can easily become impractically large as discussed above.

More distantly related are volumetric block decomposition algorithms not driven by a prescribed singularity structure and targeting other use cases, such as those discussed in section 2.3.7.4. These either lack the guarantee of obtaining a purely cuboid block structure or lack generality in terms of admissible input domains.

In the following we devise a generalization of the construction routine of the motorcycle graph, which yields a rectangular partition of parametrized surfaces, to parametrized volumes. The specific cuboidal partition obtained from this routine, due to being a volumetric cell complex rather than a graph, we term the 3D *motorcycle complex*. While point-like motorcycle particles do not extend to the 3D setting, we adopt the concept's name to reflect the close analogy in terms of

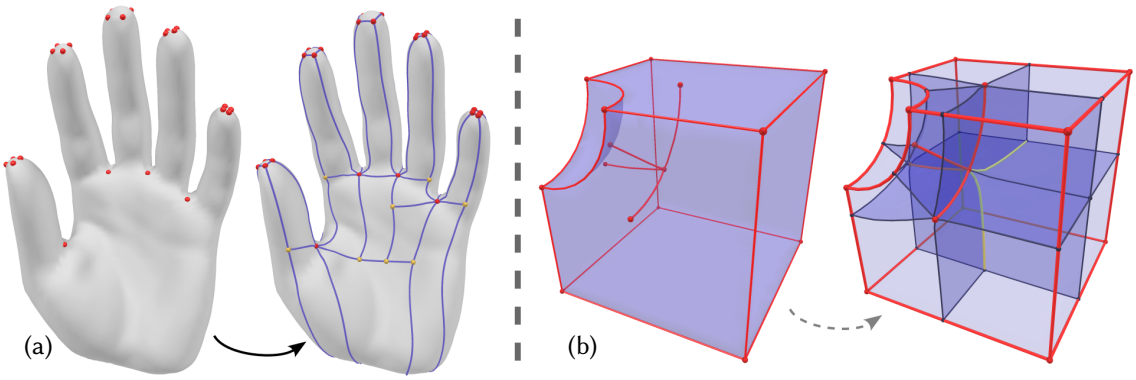


Figure 5.1: (a) Methods of obtaining a surface T-mesh by constructing the *motorcycle graph* equipped with various guiding sctructures are known [Eppstein et al. 2008; Myles et al. 2014; Campen et al. 2015]. Conceptually, starting from singularities (red) blocking iso-lines (purple) are traced along the surface, potentially meeting in T-junctions (yellow). (b) In the volumetric setting, singularities are networks of curves rather than isolated points; a solution for obtaining a volumetric T-mesh has not been described in prior literature. We generalize the motorcycle graph construction routine to volumes, specifically ones equipped with a seamless map.

the resulting partitions and their properties.

5.3 Generalizing the Problem

Metric In the following, we argue about curves or surfaces in a mesh $\mathcal{M}$ being straight, planar, or orthogonal *with respect to* ϕ or *in the* ϕ*-metric*. This is to be understood as measuring these objects' images under ϕ in $\mathbb{R}^n$—or equivalently: measuring in $\mathcal{M}$ using the metric tensor that is the pull-back through ϕ of the Euclidean metric tensor. Note that while notation in the following is based on seamless parametrizations, in principle it applies to other guiding structures as well. Specifically, it trivially extends to quad or hex meshes when assuming each quad or hex to be parametrized via the unit square or cube, respectively.

Motorcycle graph For the case of a seamless parametrization ϕ providing guidance on a surface mesh $\mathcal{M}$, the construction routine for the motorcycle graph can be summarized as follows. At each point $p_i \in \mathcal{M}$ where ϕ is singular, for each direction $\vec{d}_i^j$ of an incident iso-line of ϕ a particle $(p_i, \vec{d}_i^j)$ (the name-giving *motorcycle*) is placed. If $\mathcal{M}$ contains feature vertices, equivalent particles are placed at these; both boundaries and feature curves are considered traces ab initio. Simultaneously, each particle starts tracing (with unit speed, from p_i in direction $\vec{d}_i^j$) a curve across $\mathcal{M}$ that is straight with respect to ϕ, i.e., it is a parametric iso-curve (taking transitions into account). A particle stops when it hits: (i) a trace, (ii) a singular or feature point. Upon termination, the collection of traces forms a surface-embedded graph, the motorcycle graph, in which feature points and singularities are guaranteed to be represented as graph nodes and feature curves and boundaries are preserved as graph edges. When instead ignoring stopping criterion (i) and stopping only at the boundary or critical points, the resulting graph is the base complex. Otherwise it is only a subgraph of the base complex.

Generalization: Motorcycle Complex The idea behind the motorcycle graph does not generalize easily to higher dimensions. While in 2D *curves* (traces of particles) are sufficient to partition the two-manifold into patches, in 3D *surfaces* are required to partition the manifold into blocks. These cannot be modeled as traces of some finite number of moving point particles.

Instead, we interpret the construction process as an equivalent brush fire expansion process, in such a way that it is dimension-generic, and yields a partition of the n-dimensional domain into n-dimensional cells. Interpreting the partition as a cell complex, we call it the *motorcycle complex*. The motorcycle graph then is the two-dimensional instance of this dimension-generic cell complex and its k-cells are hyperrectangles (under ϕ), i.e., points ($k = 1$), line segments ($k = 2$) and rectangles ($k = 2$). The goal of this generalization is to devise a practicable construction routine for the *3D motorcycle complex*, such that it results in a volumetric cell complex consisting of only cuboids ($k = 3$) and the aforementioned elements.

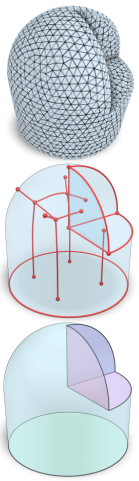
Critical entities The construction of the motorcycle graph in 2D as well as the envisioned construction in higher dimensions are based on *critical entities*, as defined below. Note that if one would only consider the *validity* of a partition, considering only singularities and boundaries would suffice. Feature preservation additionally requires treating features as critical entities.

Definition 5.1 (Critical entities in 2D). In 2D the critical entities of $(\phi, \mathcal{M})$ are any maximal, connected point sets of different dimensionality for which alignment with the integer-grid would be required for a 2D integer-grid map. Concretely, these are:

0D Critical nodes are feature and singular vertices.

1D Critical links are maximal chains of feature or boundary edges unbroken by critical nodes.

Definition 5.2 (Critical entities in 3D). We define as critical entities of $(\phi, \mathcal{M})$ any maximal, connected point sets of different dimensionality for which alignment with the integer-grid would be required for a (to-be-computed) integer-grid map. Concretely, these are:



0D Critical nodes are feature vertices and nodes of the singular graph (red spheres in the inset).

1D Critical links are maximal chains of singular or feature edges unbroken by critical nodes (red cuves in the inset).

2D Critical regions are maximal connected sets of boundary or feature facets unbroken by critical links (colored patches in the inset).

Under an input seamless map, the 1D and 2D entities are already aligned with (non-integer) iso-lines and iso-planes in parameter space.

5.4 Generalizing the Solution

In the construction of dimension-generic motorcycle complexes, conceptually, a fire is ignited simultaneously at all critical entities of the n-dimensional seamless parametrization ϕ on $\mathcal{M}$. It is confined to spread within all $(n-1)$-dimensional isoparametric submanifolds that contain the critical entities, and cannot cross points already burnt. For $(n-1)$-dimensional critical entities, this implies they can simply be considered fully burnt from the start.

For $n = 2$, singular vertices are 0-dimensional critical nodes, and the isoparametric submanifolds are 1-dimensional iso-curves with respect to ϕ. The fire of critical nodes therefore spreads along all parametric isocurves incident on them. Boundaries and feature curves are critical links, and thus fully contained in a single isocurve each; they are considered burnt ab initio. This coincides with the classical definition of the 2D motorcycle graph.

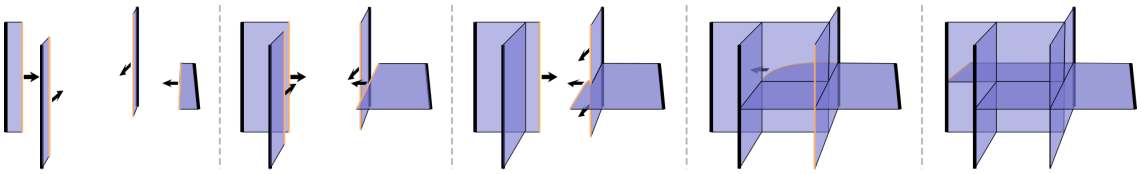


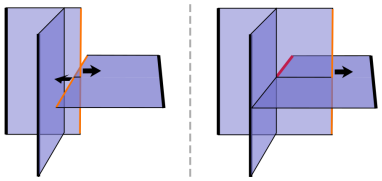
Figure 5.2: Illustration of the brush fire process in 3D. For simplicity and visual clarity only a single iso-surface per critical link (bold black) is shown, clipped to a cubical region, in a setting where iso-surfaces are planar. The fire front is highlighted in orange, its conceptual direction of expansion is indicated by arrows.

For $n = 3$, singularities are 1-dimensional critical links, and isoparametric submanifolds are 2-dimensional iso-surfaces. The fire of critical links therefore spreads along all parametric iso-surfaces they are contained in (which are bound to exist, as critical links are themselves iso-curves under ϕ). The example in 5.2 illustrates the concept. Boundaries and feature surfaces are critical surfaces, and thus fully contained in a single iso-surface each; they are considered burnt ab initio. The fire of critical nodes on the other hand spreads along all incident parametric iso-surfaces. For a (regular) feature node these are exactly three, each orthogonal to one of the parametric axes. For singular nodes, there may be more. We discuss the properties of the implied decomposition of $\mathcal{M}$ in section 5.5.2.

5.5 The 3D Motorcycle Complex

Let us point out an important difference between the 2D and the 3D instance of the dimension-generic construction routine described above: In 2D, the fire front at any time consists of a set of isolated points. Whenever such a point reaches a location already burnt, it vanishes. This gave rise to the original motorcycle metaphor. In 3D, the fire front is a set of curves (a continuum of points). Such a curve may be partially extinguished when locally blocked by burnt terrain, while the remainder proceeds (flowing around the obstacle; fig. 5.2 center). The motorcycle metaphor thus, in contrast to the confined brush fire, applies only loosely to the *process* in 3D, but we adopt the name due to the very close analogy in terms of its *results*, the partitions and their properties.

Note that considering the fire front curves as atomic entities instead, that completely stop when any part reaches burnt terrain, would not yield the desired partition properties discussed in the following. This is demonstrated in the inset: Treating the wall coming from the right as atomic leaves a dangling edge (red). No analogue of such situations exists in 2D.



5.5.1 Notation

We define the following entities:

3D *block*: maximal connected volume bounded by burnt space

2D *wall*: maximal connected parts of burnt space that lie on the same parametric iso-surface.

1D *joint*: maximal intersection curve of two or more (non-coplanar) walls.

0D *node*: intersection point of two or more (non-parallel) joints.

These 3-, 2-, 1-, and 0-dimensional cells form a partition of the domain, but do so in a non-conforming manner. Two blocks for example, may have partially overlapping sides and that

overlap makes up only a fraction of an entire wall. This is incompatible with the cell complex formalism introduced in section 2.1, where cells of the same dimensionality are only allowed to be adjacent via full lower-dimensional cells.

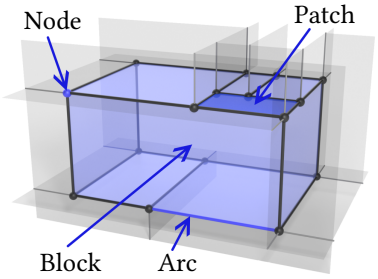
There is, however, a straightforward way of reconciling these seemingly incompatible concepts, namely by cutting walls and joints into smaller parts such that the higher-dimensional entities then do share exactly one such part. This cutting is formalized in the following way, to obtain the k-cells of a generalized cell complex:

0D *node*: nodes are induced by the above definition.

1D *arc*: maximal joint segment bounded by nodes (but without interior nodes).

2D *patch*: maximal wall region bounded by arcs (but without interior arcs).

3D *block*: maximal connected volume bounded by patches (but without interior patches).



Note that the geometric interpretation of nodes and blocks remains the same, walls and crossings are segmented into arcs and patches, respectively, and k-cells are now connected via proper bounding relations. Together, they form a cell complex $\mathcal{T} = B \cup P \cup A \cup N$ equipped with top-down incidences ∂ (and, inductively, bottom-up incidences ∂^{-1}) as defined in section 2.1.

5.5.2 Construction-Induced Properties

We call the cell complex variant obtained directly from the theoretical wall tracing routine the *raw* 3D motorcycle complex. To properly define this variant, we must first derive its properties. Most importantly, we show that the k-cells of the raw motorcycle complex are not necessarily homeomorphic to a k-ball yet. For instance, arcs may be cyclic, patches annuli or blocks solid tori. We discuss (and resolve) this in the following. Another important property, unique to the 3D case, is that the obtained block decomposition is not locally minimal; smaller partitions with the same properties are accessible via local modifications. This is addressed afterwards, in section 5.5.2.5.

Proposition 5.3 (Cells in the raw motorcycle complex). *Arcs in the raw motorcycle complex are parametrically straight, and in object space can be segments bounded by nodes* (A_1) *or closed loops* (A_2)*. Patches are parametrically planar, and in object space can be either of the following:*

(P_1) *Topological quadrilaterals bounded by four parametrically straight sides meeting at* 90° *angles in four corners.*

(P_2) *annuli, with two sides that are cyclic, parametrically straight, and mutually disconnected, or*

(P_3) *closed toroidal surfaces.*

Blocks are regular volumes under ϕ *(not containing any singularities), and in object space can be either*

(B_1) *topological cubes, with six parametrically planar sides meeting in* 90° *angles at twelve straight block edges and eight solid block corners,*

(B_2) *solid tori, with quadrilateral cross section, and four sides that are annular, parametrically planar, and meet in four* 90° *angles at four cyclic straight edges, or*

(B_3) *hollow solid tori, with annular cross section, and two disconnected sides that are parametrically planar, toroidal surfaces.*

Furthermore (B_1) *admits only* (P_1) *and* (A_1) *as elements on the block boundary.*

Notably, only cells of type (A_1), (P_1) and (B_1) represent k-balls. However, a consequence of the last aspect of the proposition is, that all cell other types can be transformed into the desired k-balls in a straightforward post-process: For all blocks of type (B_3) introduce an additional wall along the (parametrically planar) cross section; this new wall is a patch of type (P_2) that splits the block into one of type (B_2). For blocks of type (B_2) introduce an additional wall along the cross section; this new wall is a patch of type (P_1) that splits the block into type (B_1). Then, only blocks of type (B_1) remain, admitting only patches and arcs of types (P_1) and (A_1).

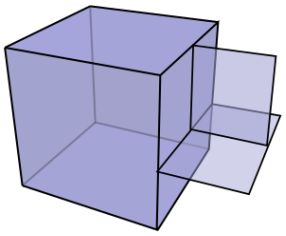
Below, we prove the above proposition via a rigorous analysis of cell geometry and topology. In the following all notions of angles, straightness and planarity are to be understood with respect to ϕ.

5.5.2.1 Cell Geometry

By construction, all singular points of ϕ are contained in arcs or nodes. The interior of each block (as well as the interior of each patch) therefore contains only regular points. This means the restriction $\phi\big|_b$ is regular, if restricted to a single block b. Consequently, the dihedral angle between any two adjacent patches incident to the same block can be at most 360° under $\phi\big|_b$, and the maximum solid angle is limited equivalently.

We first show in the following that dihedral angles formed at arcs (between two patches incident to the same block) are either 90° or 180°. Building on that, we show that the solid angle formed around nodes within a block can only correspond to a hemisphere over a plane, a sphere-quadrant over a 90° junction of two orthogonal planes or a sphere-octant over the junction of three orthogonal planes (informally: combinations of 90° and 180° dihedral angles into solid angles). Similarly, the face angles, i.e., inner angles in patches of the decomposition, are either 90° or 180°. All such angles here and in the following are to be understood w.r.t. ϕ.

Block sides First, we can observe that blocks have a boundary that is piecewise planar w.r.t. ϕ; we call each planar piece a *block side*. This is because a block is bounded by walls, which lie on parametric isoplanes under ϕ. Note that a block side may consist of one or of multiple patches from the same wall; in the inset the right block side consists of three patches, due to T-joints implied by patches incident orthogonally to the block side. Because block sides are planar, the inner dihedral angle of two patches in the same block side is 180° across the shared arc. Equivalently, at any node that lies *within* the block side, the solid angle formed by all patches incident to the node describes a hemisphere over the block side.



Block edges Next, we establish the angles at block edges, i.e., at arcs where patches from two different walls (both incident to the block) meet. Around critical links, isoparametric surfaces emanate in 90° intervals. Therefore, at these, all incident blocks have 90° edges. Away from critical links, walls end only where they hit another wall (which may be a traced one, a boundary or a feature surface). We refer to the wall that is hit as the primary wall and the other wall as the secondary wall. As both walls are iso-parametric surfaces, they must either be orthogonal or lie on the same isoplane. In case two walls meet head on within the same isoplane, they are effectively merged into the same wall; technically, they meet in a 180° angle, but no arcs are formed. Otherwise they are orthogonal and the primary wall—because it blocks the secondary wall—must have arrived first and thus, by construction, must extend to both sides of the joint curve. Hence, in both cases 90° block edges are formed at either side of the blocked wall.

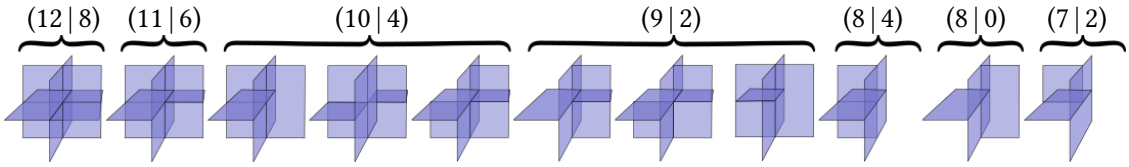
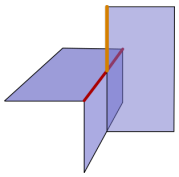


Figure 5.3: All wall configurations that may occur around a single node in a regular region (up to symmetry). The values $(n_p \mid n)$ specify the numbers of walls w and the numbers of solid corners c (all of 90° type) incident at the node. Any other configuration cannot occur in the motorcycle complex because it contains open edges or 270° edges.



In summary, any two different walls meeting along a joint curve form 90° block edges; the construction process precludes block edges of 270° or 360° (shown in the inset in red and orange, respectively). This makes sense intuitively, because in either case a wall would simply have continued to expand further as per the construction routine. Hence, at any node that lies *within* a block edge, the solid angle formed by all patches incident to the node and the block is a sphere-quadrant over the block edge.

Block Corners Nodes are formed wherever more than two walls meet in one point. At critical nodes isoparametric walls emanate in each possible direction, such that the solid angle formed between any three such walls is equivalent to a sphere-octant, i.e., a block corner. The arcs in such a corner form 90° face angles at each incident patch. Along critical links, all possible walls with wall normals orthogonal to the link alignment direction emanate, forming 90° edge sectors in between. Any further wall arriving within one of these 90° sectors, enclosed by two walls, must have a normal vector parallel to the link alignment direction. Because the interior of each sector is regular, the only way the third wall can partition the sector is into two sphere-octant block corners. In these corners, arcs form 90° face angles at incident patches. Within any sector, in which no third, orthogonal wall arrived at the same level, the created node lies on a block edge, with a sphere-quadrant solid angle. Depending on whether the sector is neighboring a bisected one, 90° or 180° face angles are formed by arcs incident at the node. As singularities are fully covered by critical nodes and links, any other nodes left to discuss are fully regular. Figure 5.3 lists all the possible patch configurations that may occur at such a regular point. Obviously, they are all subsets of the *complete* configuration labeled (12 | 8), with the maximum of 12 patches meeting in one point. In all of them, inner face angles of patches are 90° or 180°. Hypothetical configurations that exhibit 270° or 360° block edges have been excluded, because those can not occur, as argued above.

5.5.2.2 Cell Topology

Taking into account the restricted nature of face angles, dihedral angles and solid angles at cells of the complex, we can reason about the topology of created cells. For this we employ the Gauss-Bonnet theorem for 2-manifolds M (potentially with boundary):

$$\int_M K \, dA + \int_{\partial M} k_g \, ds = 2\pi\chi(M) \tag{5.1}$$

The first term integrates the Gaussian curvature K over any point in the surface, the second term integrates only over the geodesic curvature k_g of the boundary ∂M, and $\chi(M)$ is the Euler characteristic of M, given as:

$$\chi = 2 - 2g - n_b, \tag{5.2}$$

where $g \geq 0$ is the genus of M (informally: the number of handles) and $n_b \geq 0$ the number of separate boundary components of M. In our discrete setting, where any surface (and its boundaries) are piecewise planar (straight) eq. (5.1) can be discretized to sum over angle defects at nodes n:

$$\sum_{n \in M \setminus \partial M} \Omega_n + \sum_{n \in \partial M} \Theta_n = 2\pi\chi(M), \tag{5.3}$$

where Ω_n is the inner angle defect, i.e., 360° minus the sum over face angles incident on n, and Θ_n is the boundary angle defect, i.e., 180° minus the sum over incident face angles.

Arc types Arcs can either be true segments bounded by two different nodes, loops bounded by the same node on either side (both considered (A_1)), or even true loops with no nodes on them (A_2). Neither the construction routine nor the above geometric considerations narrow this down any further. The straight sides of patches, being sequences of arcs on the same isoline and (potentially) bounded by patch corner nodes on either side, exhibit the same topological variations as individual arcs. The equivalent is true for the straight edges of blocks.

Patch types Patches are completely planar in their interior and the arcs bounding them form only 90° or 180° inner angles. Hence, angle defects occur at 90° patch corners, and amount to 90° each. Plugging this into eq. (5.3) yields:

$$\frac{n_{90°}}{4} = \chi = 2 - 2g - n_b, \tag{5.4}$$

where $n_{90°}$ is the number of 90° patch corners, of which there may be 0 (nothing in the construction routine or the above considerations excludes this case) or more. The only valid solutions $(n_{90°}, g, n_b)$ to this are $(4, 0, 1)$, $(0, 0, 2)$ and $(0, 1, 0)$. These are a rectangle (four corners and a single boundary, (P_1)), an annulus (formed by gluing together opposite sides of a rectangle just once, (P_2)) or a torus surface (formed by gluing together both opposite sides of the rectangle, (P_3)). Notably, the rectangle case (P_1) admits only arcs of type (A_1).

The next question to answer is, which block types are bounded by such patches. Block sides are connected sets of coplanar patches, delimited by straight block edges that can meet only in block corners and only in 90° angles, inducing angle defects of 90°. Hence, block sides follow the same pattern as patches and exhibit the same three topological variations. Note that a block can have either rectangular or annulus or toroidal sides, but not a mixture of these. This is because torus sides are unbounded and hence can not join with any other side types, and rectangular sides have 90° angles at their corner nodes, which would induce similar corners in adjacent sides—incompatible with annuli.

Block types Albeit blocks being volumetric 3-manifolds, we can characterize them via their boundary.

Single boundary component Assume first the boundary of a block consists of exactly one component. The block is a 3-manifold, hence its boundary is a single closed 2-manifold surface and can be studied via the Gauss-Bonnet theorem. Angle defects on the block boundary are concentrated at corner nodes, of which (as discussed above) only ones with sphere-octant solid angles can exist, for which the angle defect is 90° each. Plugging this into eq. (5.3) yields:

$$n_{8\text{th}}/4 = \chi = 2 - 2g, \tag{5.5}$$

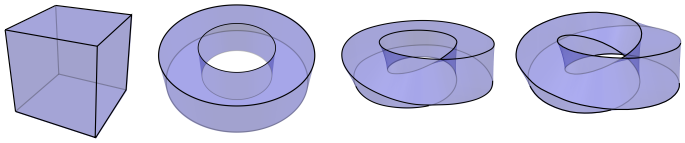
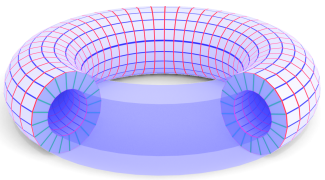


Figure 5.4: Blocks of the raw motorcycle complex with connected boundary can only be cuboidal or toroidal. Toroidal blocks may have 4, 1, or 2 annulus sides depending on their *twist*.

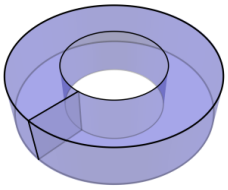
where $n_{8\text{th}}$ refers to the number of sphere-octant block corner nodes. The only valid solutions $(n_{8\text{th}}, g)$ to this are $(8, 0)$ and $(0, 1)$. The first solution is a block with genus $g = 0$ and eight corners, which admits only rectangular patches of type (P_1). Employing Euler's polyhedral formula $V - E + F = 2 - 2g$ with $V = 8$ and rectangular sides $(E = 2F)$ implies that it must be a cuboid with 8 corners, 6 sides and 12 edges. The latter solution is a block with genus 1 and no corners, i.e., a torus with annulus sides (rectangle sides would form corners). There is an infinite number of structurally different such tori: The parametric *twist* of the torus can be an arbitrary number k of quarter turns. The twist $k90°$ is observable as the cumulative transitional rotation of the parametrization in a cycle around the solid torus handle. It is a k-times multiple of 90° rotations around the parametric axis aligned with the torus handle. The block has 4 sides if $k \bmod 4 = 0$, 2 sides if $k \bmod 4 = 2$, and only 1 side if k is odd.

Multiple boundary components Assume now, that a block has more than a single boundary component. Each boundary component is piecewise planar, and different pieces form only convex angles. As the block interior is regular, strictly convex angles would imply that the boundary either does not exhibit multiple disconnected components, contradicting the assumption, or the block is not a single connected volume, contradicting the definition of blocks. Hence boundary components can only be coplanar, in which case there must be exactly two closed boundary components, as more again would not result in a single connected volume. Because we established above that block sides come in the same topological variations as patches, the block must be bounded by two closed toroidal surfaces. The shape resulting from this is a hollowed solid torus, displayed in the inset.



5.5.2.3 Recovery of n-Balls

As noted above blocks may be cuboids (B_1), solid tori (B_2), or hollowed tori (B_3), with arcs and patches of type (A_2), (P_2) or (P_3) only occurring as bounds of the latter two block types. We observe that blocks of type (B_2) and (B_3) are conceptually obtained by gluing together two or four pairwise opposite sides of a cuboid, respectively. An equivalent gluing is applied to patches and arcs on these block's boundaries. Hence, for all of these the gluing can be reversed, and the cells transformed into (self-adjacent) n-balls by introducing one or two additional isoparametric walls per block. For toroidal blocks a single wall representing the quadrilateral cross-section suffices, as shown in the inset. For the hollow torus, the first introduced wall represents an annular cross-section and cuts it into a self-adjacent solid torus. A second, orthogonal wall can split this solid torus as described above. Using this modification, a pure cuboid block complex is obtained in any case, avoiding the need for further special case handling in subsequent operations. Topologically, its n-cells are open n-balls, geometrically they are n-dimensional *hyperrectangles* under ϕ. Note that due to self-adjacency, it is not necessarily true that the bounds of an n-cell together form an n-sphere.



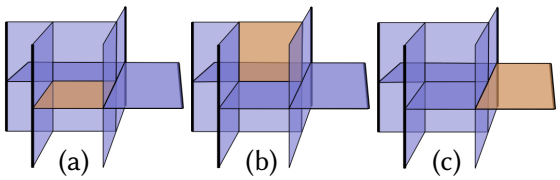


Figure 5.5: Illustration of *reducibility*, based on the example from fig. 5.2. The orange wall is (a) regular-removable, due to yielding a new cuboid after removal, (b) not removable, due to yielding a non-cuboid, and (c) removable but not regular-removable, due to being incident to a critical arc.

5.5.2.4 Critical Entity Preservation

By construction, it is ensured that any critical entities of the input remain represented as discrete elements in the cell complex. At critical nodes in the input, walls emanate in each possible direction, of which there are at least three, ensuring they form a node of the 3D motorcycle complex, coinciding with the critical node. At critical links in the input, walls also emanate in each possible directions, of which there are at least two, which along their intersection form a sequence of arcs of the complex, coinciding exactly with the critical link. Critical regions are considered burnt ab initio, i.e., they immediately become walls, hence are represented as sets of connected patches in the later complex. We call the corresponding cells of the motorcycle complex critical nodes, critical arcs and critical patches.

5.5.2.5 Reducibility

Just as in the 2D case (section 5.1.2), we cannot expect the resulting (raw) motorcycle complex to describe a globally minimal (i.e., smallest) partition with the desired properties (cuboidal, regular, aligned). It is worthwhile considering the aspect of local minimality, though. To this end we define the following:

Definition 5.4 (Reduction). A reduction is the operation of merging two adjacent (hyperrectangular) n-cells of the motorcycle complex into one n-cell by removing exactly a single $(n-1)$-cell shared between these, such that this larger cell is also a hyperrectangle. Any $(n-1)$-cell for which such a reduction exists, we call *removable*. A complex without removable cells, we call *irreducible*.

We furthermore define a restricted notion of regular-reductions. For this notion, we consider any critical patches, critical arcs and critical nodes to never be removable by reduction. Any reduction that is still admissible is a *regular-reduction*. A complex that admits no regular-reduction is *regular-irreducible*. This notion is relevant for use cases (such as in section 6.3) that require singularities to lie only at block edges, not in the interior of block facets.

Under a general position assumption on the location of singularities, the standard 2D motorcycle graph (while reducible) is regular-irreducible. Only when singularities are aligned under the parametrization, such that motorcycles meet in a frontal manner there may be options for regular-reduction. As demonstrated in fig. 5.5, the situation is different in the 3D case: Even regular-irreducibility is not a given, the brush fire result commonly contains regular-removable walls. The underlying reason is related to the discussion at the beginning of section 5.5: While motorcycles in 2D are points, and collisions with traces are isolated instantaneous events, in 3D the more complex brush fire that forms a wall may stop in one place while proceeding in another.

Wall Retraction We therefore propose to subsequently reduce the result of the brush fire process to a locally minimal, i.e., either regular-irreducible or irreducible state, as desired. To this end, we perform *wall retraction*: (regular-)removable walls are greedily removed, ordered

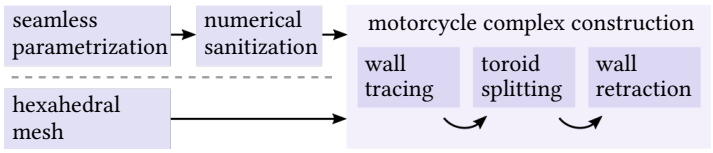


Figure 5.6: We take as input either a seamless parametri-zation (on a tet mesh) or a hexahedral mesh, and compute an induced motorcycle complex in three algorithmic steps.

by their parametric distance from their origin. Intuitively, this can be interpreted as retracting fire walls in places where they have spread unnecessarily far in the brush fire expansion. Any lower-dimensional entities becoming (regular-)removable in the process are cleaned up as well. It would be conceptually attractive to avoid this redundancy already during the expansion process, but this is not straightforward. Practically, the overhead due to the reduction happening after the fact is benign (see experiments in section 5.7).

Where distinction is necessary, we refer to the non-reduced brush fire result as *raw motorcycle complex*, while *motorcycle complex* is generally meant to refer to the reduced version (with additional walls inserted in rare toroidal cells, theorem 5.3). In section 5.6 we describe the construction as well as the reduction process in detail.

Remark (Base Complex Reduction) One could start from the base complex and apply reductions until an irreducible minimum is achieved. However, the base complex can be very large, hampering practical construction, and, according to our experiments reported in section 5.7, retraction starting from the base complex commonly ends up in worse local minima, i.e., complexes of larger size.

Remark (Sparse Serial Construction) In the 2D case, a not only regular-irreducible but fully irreducible motorcycle graph can be obtained right away by not tracing motorcycles simultaneously but serially, and omitting motorcycles whose neighbors around a singularity have already been traced [Eppstein et al. 2008, §7]. A similar strategy can be applied in the 3D setting as well, as we detail in section 5.7.3. However, the reported experiments show that this serial strategy commonly leads to more complex results than wall-retraction applied to the standard simultaneous strategy; we therefore focus on the latter in the following.

5.6 Construction in Practice

We describe two implementations, one to compute a motorcycle complex of a hexahedral mesh (primarily as an intuitive entry) and one to compute a motorcycle complex of a seamless parametrization on a tetrahedral mesh. This is summarized in fig. 5.6. In the former case we can exploit that all relevant isosurfaces are available explicitly as facets of hexahedral elements, resulting in a discrete (combinatorial rather than geometric) algorithm; in the latter case isosurfaces arbitrarily cross mesh elements in a continuous manner, requiring additional efforts. Figure 5.7 illustrates the algorithm on two example models. A video showcasing various aspects of the construction routine on example models has been published in the supplemental material of [Brückler et al. 2022b].

5.6.1 On Hexahedral Meshes

In this case the input is a hexahedral mesh $\mathcal{H} = H \cup F \cup E \cup V$, consisting of hexes, facets, edges and vertices. Implicitly, it has a natural seamless parametrization, mapping hexes to unit cubes. For a regular edge e and an incident facet $f \in \partial^{-1} e$ let $\mathrm{opp}(f, e)$ denote the facet incident to

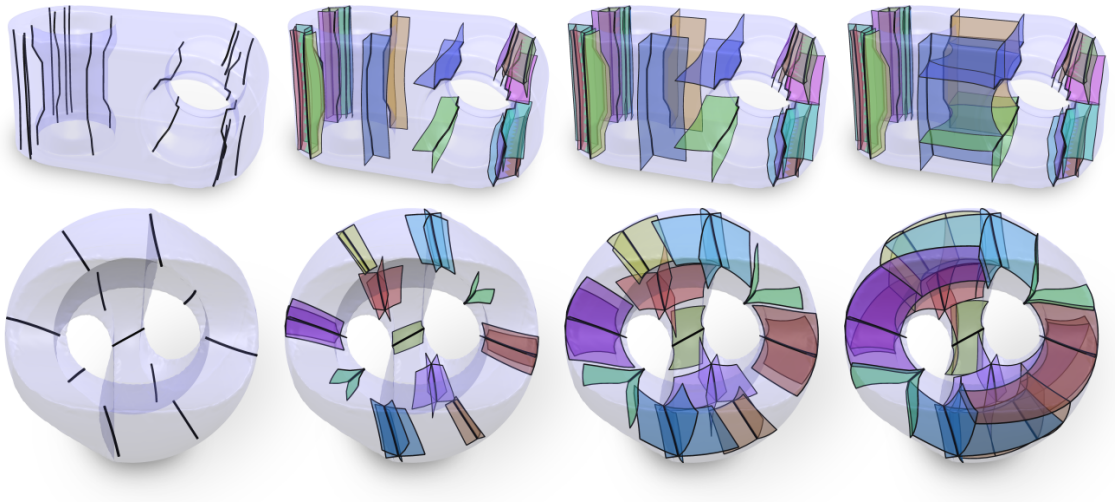


Figure 5.7: Snapshots of the motorcycle complex construction algorithm (left to right) in seamless parametrizations of two example objects. The black curves in the second column are the parametrization's interior singularities, which spawn the fire walls that partition the object's interior. Wall facets are rendered transparently, and other facets omitted. For visual clarity, fire fronts are shown as smooth curves here.

Algorithm 5.1: MOTORCYCLE COMPLEX OF A HEX MESH

```
   // Igniting //
   foreach edge e on a critical link or incident to a critical node do
   |  foreach facet f ∈ ∂⁻¹e do
   |  |  Q.push(e, f, 0)                                  // ignite fire front
   // Fire spreading //
   foreach facet f on a critical surface do tag f                    // pre-burn
   while Q non-empty do
   |  (e, f, d) ← Q.pop()
1  |  if uncrossed(e) and untagged(f) then          // stop at burnt terrain
   |  |  tag f                                                       // burn
   |  |  foreach uncrossed non-critical edge e′ ∈ ∂f do
   |  |  |  if opp(e′, f) is not tagged then
   |  |  |  |  Q.push(e′, opp(e′, f), d + 1)                // propagate front
```

e not incident to a common hex with f; for a boundary edge it may not exist. For an edge e, $F_e = \partial^{-1} e \setminus \partial\mathcal{H}$ denotes the set of incident interior facets.

The algorithm makes use of a priority queue Q of (f, d) tuples, each with an edge e, a facet f, and a distance $d \in \mathbb{N}$. Queue elements are ordered by d, smallest first. Due to the hexes being parametrized as unit cubes, distances are incremented by exactly 1 on each fire propagation; this is equivalent to a breadth-first expansion using a (non-priority) queue. A priority queue is still employed here for a straightforward extension to unquantized settings such as a parametrized tet mesh.

The condition uncrossed(e) (algorithm 5.1) is true iff at most two facets incident on e are tagged as burnt or e is critical or incident to a critical node. This means that an edge that has already been crossed will not be crossed again (in orthogonal direction). This implies that ties (two fire walls reaching an edge orthogonally with the same distance d) are broken arbitrarily. Note that even if the tie is broken differently on neighboring edges, the resulting partition will be structurally valid (as if both fire fronts had continued), i.e., there is no need for global coordination.

Once the algorithm terminates, the union of all tagged facets form the walls of the raw motorcycle complex, partitioning the hexahedral mesh into blocks B_i, each consisting of $m_i \times n_i \times o_i$

hexes for some $m_i, n_i, o_i \in \mathbb{N}$. The explicit structure and connectivity of the motorcycle complex is then easily discovered by exploiting the connectivity of the underlying hexahedral mesh.

5.6.2 On Seamless Parametrizations

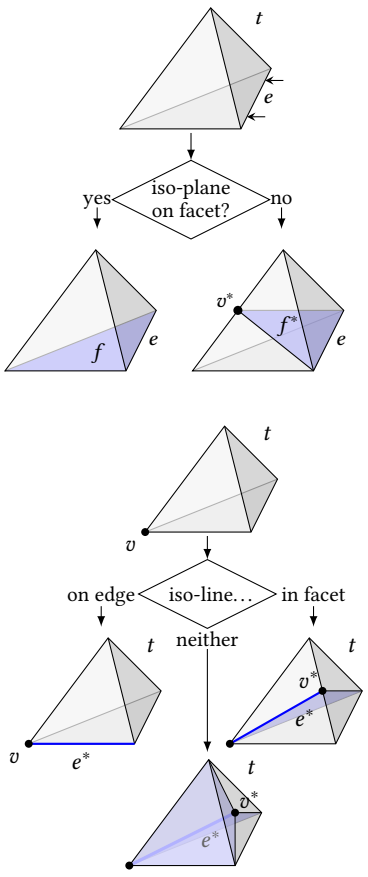
Here the input is a tet mesh $\mathcal{M} = T \cup F \cup E \cup V$, consisting of tetrahedra, facets, edges and vertices, equipped with a seamless parametrization ϕ with tet-wise charts ϕ_t expressed via vertex parameters $\boldsymbol{u}_t^v$. In contrast to the above algorithm, here we cannot simply walk along the faces of the mesh: The isosurfaces relevant for the motorcycle complex do not coincide with the tetrahedral mesh's facets, but cross its tets arbitrarily. We thus need to perform the brush fire expansion through the interior of tets. Inside each tet the situation can furthermore be highly complex, with multiple fire walls meeting in arbitrary configurations; essentially, within each tet a separate Euclidean 3D motorcycle complex problem is to be dealt with.

We can simplify implementation significantly by refining the mesh on the fly while spreading the fire, so as to have it coincide with facets of the mesh. This simplifies not only the propagation process, but also the representation of the motorcycle complex and the final discovery of its structure and connectivity. Algorithm 5.2 spells out this process—in close analogy to algorithm 5.1, but extended to perform and deal with the refinement of the mesh.

For the following, let $v_i(e)$ with $i \in \{0, 1\}$ denote either of $v \in \partial e$, and $\phi_t(t)$ the tet t in parameter space. Furthermore, ISOLINE$(\boldsymbol{u}, \vec{l})$ is the line based at $\boldsymbol{u}$ and extending along the unit parametric axis vector $\vec{l}$. Equivalently, ISOSURFACE$(\boldsymbol{u}, \vec{n})$ is the plane based at $\boldsymbol{u}$ with principal axis-aligned normal $\vec{n}$.

5.6.2.1 Mesh Refinement

The method GETORMAKEISOFACET$(t, e, \vec{n})$ (algorithm 5.2) performs the following: If a parametric facet $f \in \partial^{-1} e \cap \partial t$ lies within the iso-plane with normal $\vec{n}$ containing e, then that facet is returned. Otherwise, the iso-plane partitions $\phi_t(t)$ into two tetrahedral parts and intersects the edge $e' \in \partial^2 t$ opposite of e (for which $\partial e \cap \partial e' = \varnothing$). In this case, a new vertex v^* is inserted along e', with coordinates $\boldsymbol{u}_t^{v^*} = \alpha \boldsymbol{u}_t^{v_0(e)} + (1 - \alpha)\boldsymbol{u}_t^{v_1(e)}$ chosen such that v^* lies on the iso-plane. The edge e' and any incident higher-dimensional elements are then split by this new vertex (see section 2.2.2). Object and parametric coordinates in all surrounding tets are interpolated with the same α to ensure consistency. Afterwards, the iso-plane coincides with the new facet f^* connecting e and $v*$; this facet is returned.



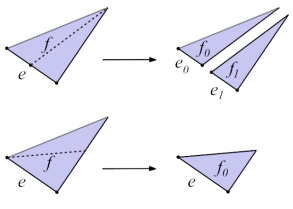
Equivalently, GETORMAKEISOEDGE$(t, v, \vec{l})$ (algorithm 5.2) tries to retrieve an edge $e \in \partial^2 t$ that lies within the iso-line with direction $\vec{l}$ based at v. If such an edge exists, it is returned. Otherwise the iso-line might lie within a facet $f \in \partial t \cap \partial^{-2} v$, piercing the edge $e \in \partial f$ opposite of v within that facet (for which $v \notin \partial e$). In that case a new vertex v^* is inserted along e and the edge is split in the same fashion as explained above. The new edge e^* connecting v to v^* then lies on the iso-line, and is returned. If the iso-line neither lies within an edge nor a facet, it must pierce the facet $f' \in \partial t$ opposite of v (for which $v \notin \partial^2 f'$).

Algorithm 5.2: MOTORCYCLE COMPLEX OF A SEAMLESS PARAMETRIZATION

```
// Igniting //
E_iso = critical link edges
foreach critical node vertex v do
    foreach tet t incident on n do
        foreach principal parametric direction l⃗ do
            if isoLine(u_t^v, l⃗) intersects φ_t(t) then
1               E_iso ← E_iso ∪ getOrMakeIsoEdge(t, v, l⃗)
foreach iso-edge e ∈ E_iso do
    foreach tet t incident on e do
        foreach principal direction d⃗ ⊥ (u_t^{v2(e)} − u_t^{v1(e)}) do      // d⃗: expansion direction
            n⃗ ← d⃗ × (u_t^{v2(e)} − u_t^{v1(e)})                            // iso-surface normal
            if isoSurface(u_t^{v0(e)}, n⃗) intersects φ_t(t) then
2               f ← getOrMakeIsoFacet(t, e, n⃗)
                Q.push(e, f, 0, d⃗, n⃗)                                    // ignite fire front
// Fire spreading //
foreach facet f on a critical surface do tag f                              // pre-burn
while Q non-empty do
    (e, f, d, d⃗, n⃗) ← Q.pop()
    if uncrossed(e) and untagged(f) then                                    // not crossing burnt terrain
        tag f                                                               // mark facet as burnt
        foreach uncrossed edge e′ ∈ ∂f \ E_iso do
3           d′ ← d + extent(e, e′, d⃗)
            foreach tet t′ incident on e′ do
4               τ ← transitionFromTo(f, t′)
                if isoSurface(φ_{t′}(v0(e′)), τn⃗) intersects φ_{t′}(t′) then
                    f′ ← getOrMakeIsoFacet(t′, e′, τn⃗)
                    if untagged(f′) then
                        Q.push(e′, f′, d′, τd⃗, τn⃗)                          // propagate front
                        break
```

In this case, a new vertex v^* is inserted within f', with coordinates $\boldsymbol{u}_t^{v^*}$ retrieved as barycentric interpolations of the coordinates of vertices incident on f', such that it lies on the iso-line. A facet split of f' at v^* is performed, and new elements inserted to maintain a pure tet mesh. Object and parametric coordinates in all surrounding tets are interpolated with the same barycentric coordinates to ensure consistency. The new edge e^* connecting v to v^* then lies on the iso-line, and is returned. All cases are illustrated in the inset. In our implementation, publicly available as the open source library MC3D[1], numerical robustness is ensured by representing split vertex coordinates exactly as rational numbers using the GMP library. To that end, it requires *exactly* seamless parametrizations as produced by our sanitization library TRULYSEAMLESS3D described in chapter 4.

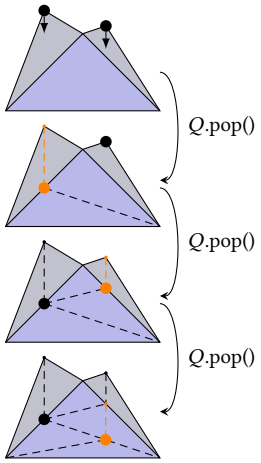
Queue updates Additionally, when splits are perfomed in to obtain iso-facets, edge and facet references in the queue might be invalidated and need to be updated. When an edge e is split, each queue entry (e, f) needs to be replaced by two entries (e_0, f_0) and (e_1, f_0), with sub-edges e_0, e_1 and sub-facets f_0, f_1. When a facet f is split, but not the edge e of an entry (e, f), it is replaced by (e, f_0), where f_0 is the sub-facet



[1] https://github.com/HendrikBrueckler/MC3D

incident on e. A more efficient (slightly more involved) implementation alternative is to postpone these queue updates: We keep a binary forest that records the facet split hierarchy: for each facet that gets split, a record of the two resulting sub-facets is kept. When an entry with facet f and edge e is popped from the queue but f does not exist in the mesh anymore (because it was split), we look up its two children in the hierarchy. Either one or two of these has an edge that is a sub-edge of e. We push the children *with* an e-sub-edge into the queue and continue. This may proceed recursively, until the sub-elements currently present in the mesh are reached.

Concurrent splitting A further modification over algorithm 5.1 is necessary for algorithm 5.2: The queue is ordered by d only secondarily; primarily, queue entries with e not lying in an original mesh facet are given priority. This ensures that once the brush fire front has entered the space of an original tet, it (atomically) proceeds through this space entirely (i.e., through all refinement-induced sub-tets). This prevents potentially infinite alternating split sequences that could occur when multiple fire walls are spreading inside the same original tet. The inset shows a cross-section view of such a situation. Active fire fronts (bold black circle), advance in the same direction. The one with highest priority (orange) advances next, after being retrieved from the priority queue. Giving priority in the bottom snapshot to the fire wall that has travelled the least distance (the left one, here) creates an infinite loop of concurrent splitting. Instead giving priority to the right wall until it leaves the original tet (blue) resolves this. Effectively, this ensures a finite minimum progress of each wall before it is interrupted by another, thereby preventing infinite mutual interruptions (and infinite refinement) between walls racing head-to-head.



5.6.2.2 Distance Tracking

Compared to the algorithm in section 5.6.1, in which propagation distances d can be increased in unit steps per hex, here we proceed differently to reduce mesh dependency and ensure that walls spread at unit speed with respect to the ϕ-metric. For this, we store not only the distance d but also the (unit, principal-axis) propagation direction $\vec{d}$ for each wall facet f. This allows to compute the orthogonal distance of the fire front from its origin, as intended. Let t be the tet incident to f with respect to whose chart ϕ_t the propagation direction $\vec{d}$ is defined. For simple computation, we assume that the distance $d_{\vec{d}}(e)$ of a fire front edge e from its origin along $\vec{d}(e)$ is always the minimum distance of any point along the edge $d_{\vec{d}}(e) = \min(\vec{d}^{\mathsf{T}}\boldsymbol{u}_t^{v_1(e)}, \vec{d}^{\mathsf{T}}\boldsymbol{u}_t^{v_1(e)}) + c$, with c being a shift that is constant within ϕ_t. Then the function extent(f, e, e', n) (algorithm 5.2) is easily defined as extent$(e, e', n) = d_{\vec{d}}(e') - d_{\vec{d}}(e)$. During propagation both the expansion direction $\vec{d}$ and the wall normal $\vec{n}$ are transformed using the chart transition between the previous tet t and the next tet t' (algorithm 5.2).

Note that with this notion of distance, lateral propagation (orthogonal to n) is associated with no increase in distance; this means a fire front that is partially blocked (as in fig. 5.2 center) laterally flows around the obstructing wall in a virtually instantaneous manner, rather than forming the circular front conceptually depicted in fig. 5.2 center right.

Also note that this implementation performs propagation in a facet-by-facet manner, i.e., fire front collisions are not handled in a continuous manner. In particular, this can lead to non-minimal results. But as non-minimality is an inherent property either way (section 5.5.2.5), and as we are

therefore going to reduce the resulting complex anyway, the significant added complexity of a continuous collision resolution would be unlikely to be justified in practice.

5.6.2.3 Post-Processing

Torus splitting In order to turn occasional blocks of type (B_2) (cf. theorem 5.3) into cuboid blocks, one simply detects these (by counting corners) and starts a new brush fire inside the block from an arbitrary point on one of its (cyclic) arcs, confined to the iso-plane that is orthogonal to the two walls incident at that point. This yields an additional wall, cutting the toroidal block to a (self-adjacent) cuboid block. For the unique case of the hollow torus (B_3) (detectable a priori as the only possible domain without any singularities), we choose an arbitrary vertex on the domain boundary, using GETORMAKEISOEDGE$(t, v, \vec{l})$ to create an iso-edge orthogonal to the boundary and spawn two orthogonal walls from this iso-edge. This yields a single (self-adjacent) cuboid block. Optionally, if self-adjacency is to be avoided, additional walls are generated in the same manner that split self-adjacent blocks into two (or more) multiply-adjacent blocks; additional walls can likewise resolve multi-adjacency of cells, if desired.

Reduction Due to the motorcycle complex typically being locally reducible, some n-cells can be merged by removing $(n-1)$-cells shared between them Critical nodes, arcs or patches may never be removed. Optionally, arcs and patches incident to critical nodes, as well as patches incident to critical arcs may also be prevented from being removed (applying only *regular-reductions*). Following theorem 5.4, an $(n-1)$-cell can be removed, if it bounds exactly two n-cells and the union of these two n cells is also a hyperrectangle (a straight arc, a rectangular patch or a cuboid block, respectively). This is easily determined:

- A node is removable, if exactly two distinct arcs coincide on it, and these arcs connect in a 180° angle in any of the incident blocks.
- An arc is removable, if exactly two distinct patches coincide on it, and these patches form a 180° dihedral angle in any of the incident blocks.
- A patch is removable, if exactly two distinct blocks coincide on it, and the patch fills a complete block side in each of these.

Removable walls are queued up, sorted by parametric distance to their origin. This distance is available as value d per facet f during algorithm 5.2 and stored accordingly. A wall's distance is defined as the minimum over its facets. We then greedily remove removable walls, starting with the farthest. Lower-dimensional cells becoming removable in the process are removed immediately, as that may result in patches becoming removable. For instance, an arc being removed merges two patches into one, and the resulting patch might be removable (only single elements can be removed at once). The removability status of all incident walls is retested after any removal and the queue updated accordingly.

5.7 Results: Motorcycle Complexes

In the following we evaluate the characteristics of the motorcycle complex, in particular in comparison to the base complex, as well as the proposed algorithms (mesh-based and parametrization-based) for its construction.

5.7.1 Mesh-Based Algorithm

We take a dataset of 261 all-hex meshes, collected by [Bracci et al. 2019], generated by a variety of hexahedral meshing approaches, e.g., [Gregson et al. 2011; Li et al. 2012; Livesu et al. 2015; Fang et al. 2016], and apply our mesh-based algorithm (section 5.6.1). Table 5.1 provides statistics on the results, including the motorcycle partitions' size before and after reduction (section 5.5.2.5). The full table has been published in the supplementary material of [Brückler et al. 2022b].

An interesting comparison is with respect to the standard base complex. We include the corresponding statistics in table 5.1. As can be observed, the motorcycle complex is often simpler by a large factor (here up to 140×). Figure 5.8 shows both the base complex and motorcycle complex on an example model, constructed using the mesh-based algorithm. Besides having an obvious positive effect on construction cost, the motorcycle complex offers benefits on the application side, as is demonstrated in chapter 6. Splitting of toroidal blocks (section 5.6.2.3) occurred in 13 of the models, a total of 48 times.

5.7.2 Parametrization-Based Algorithm

We apply the parametrization-based algorithm (section 5.6.2) to seamless parametrizations on tetrahedral meshes, generated using frame field guided parametrization, i.e., by numerical optimization of eq. (3.5) (without integer constraints, and with transition rotations prescribed by a frame field). In this we use frame fields provided by the authors of [Liu et al. 2018], corresponding to the results shown in that article. Numerical sanitization of these parametrizations (chapter 4) took less than a second for most cases, 7s for the most complex case (with 163K tets). Not employing this sanitization resulted in structural failures during tracing in *all* cases. This emphasizes the importance of performing the sanitization step.

Model	**BC**	BC^-	raw	MC^+	$\frac{MC^+}{BC}$	T	**MC**	$\frac{MC}{BC}$
EXAMPLE 3	406136	67828	9087	5780	1.4%	42%	2877	0.7%
EXAMPLE 1	74331	11385	3137	2248	3.0%	15%	1123	1.5%
EXAMPLE 2	3253	678	233	195	6.0%	14%	87	2.7%
DRAGON-HEX	12488	2959	979	724	5.8%	36%	357	2.9%
GARGOYLE	7563	1967	720	546	7.2%	38%	257	3.4%
ANC101 A1	12336	3118	1359	846	6.9%	45%	460	3.7%
FERTILITY-HEX	2002	548	221	189	9.4%	27%	76	3.8%
PEGASUS-HEX	9745	2415	1035	729	7.5%	36%	374	3.8%
KISS HEX	5019	1194	543	385	7.7%	39%	200	4.0%
ANC101	5009	1283	609	347	6.9%	39%	207	4.1%
IMPELLER STRESSTEST	878	176	184	124	14.1%	24%	37	4.2%
ARMADILLO HEX-A	5960	1491	680	516	8.7%	34%	266	4.5%
ARMADILLO HEX-B	3265	820	396	296	9.1%	31%	147	4.5%
			⋮					
EXAMPLE 5	1	1	1	1	100%	0%	1	100%

Table 5.1: Statistics on a dataset of hexahedral meshes. Reported are numbers of blocks in the base complex (BC), reduced base complex (BC^-), raw motorcycle complex (raw), regular-reduced motorcycle complex (MC^+), and fully reduced motorcycle complex (MC)—ordered by complexity of MC relative to BC. It can be observed that the raw MC is typically larger than the MC^+ (or MC) by a factor of around 1.4 (or 2.9) only, i.e., construction overhead over a hypothetical direct construction of the final MC is benign. Furthermore, notice that the fully reduced BC^- is generally significantly larger than the fully reduced MC (see the remark in section 5.5.2.5). On average one third of the MC's arcs locally form T-junctions (T).

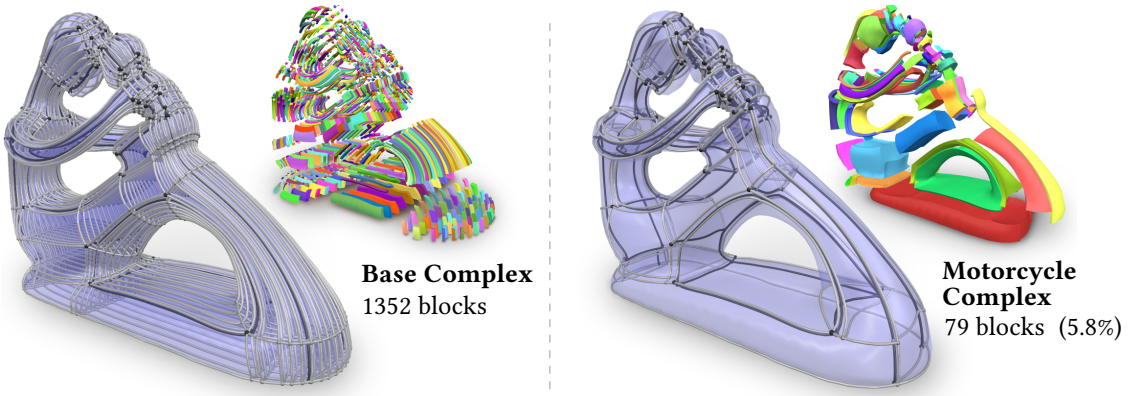


Figure 5.8: Base complex (left) and our 3D motorcycle complex (right) induced by the same seamless map, both providing a partition into cuboid blocks. The transparent views show arcs and nodes of the partition (gray: standard, black: critical), the exploded view shows individual blocks. The motorcycle complex, due to being non-conforming, partitions the object's interior into a much smaller number of blocks.

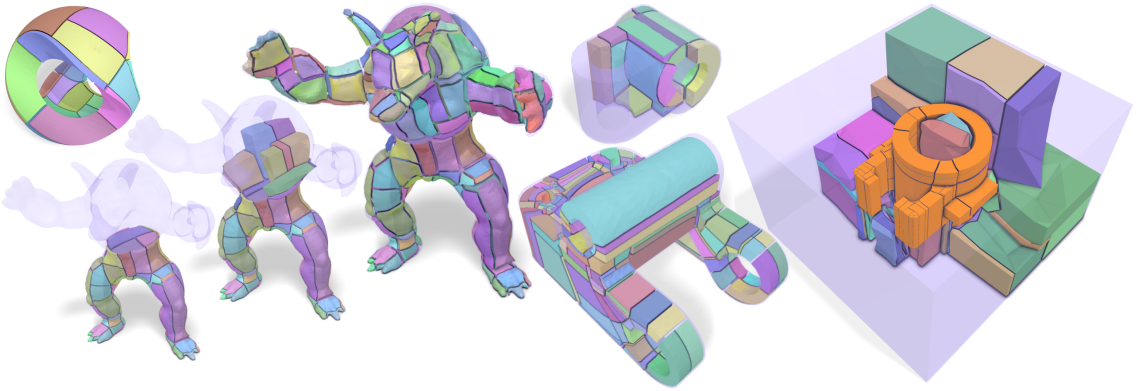


Figure 5.9: Multiple motorcycle complexes computed by our parametrization-based method. The rightmost example demonstrates exact representation of features (orange), in this case embedded within a volume.

Table 5.2 shows details about the motorcycle complex construction, including the number of facets traversed and forming the motorcycle complex walls. Notice that again the base complex is significantly more complex; the number of blocks is up to 30× higher, construction time up to 13×, memory consumption up to 9×. Figure 5.9 shows several motorcycle complexes, computed on models from the above dataset (left four models) and the dataset introduced in chapter 9.

Seamless parametrization failures The motorcycle complex is well-defined only for *valid* seamless parametrizations in general. Their fully robust generation in 3D is a problem under broad investigation, as discussed in section 3.7. In particular because our algorithm operates on generic *seamless* rather than *integer-grid* parametrizations, it was comparatively easy, though, to obtain valid input parametrizations for 15 out of 19 models from [Liu et al. 2018] already with the above simple best-effort approach following [Nieser et al. 2011]. For the remaining 4 models the obtained parametrization was not injective; due to this invalidity, our algorithm can not be applied.

5.7.3 Sparse Serial Motorcycle Complex

Let us expand on the remark in section 5.5.2.5 about the alternative of a serial motorcycle complex construction. A proposal in [Eppstein et al. 2008, §7] for the 2D case is to trace motorcycles in a

Model	tets	**BC**	**MC**	$\frac{\text{MC}}{\text{BC}}$	facets	trace	build	reduce
ROCKERARM	97 K	2446	78	3%	154 K	27.9 s	11.1 s	1.9 s
ARMADILLO	163 K	3110	132	4%	260 K	46.4 s	16.8 s	1.6 s
JOINT	42 K	205	15	7%	56 K	9.5 s	3.6 s	0.6 s
KITTEN	30 K	208	19	9%	49 K	8.3 s	3.4 s	0.7 s
BROKEN BULLET	20 K	44	5	11%	13 K	2.1 s	0.9 s	0.1 s
SCULPTURE	20 K	108	13	12%	17 K	2.8 s	1.2 s	0.1 s
FANDISK	46 K	128	19	15%	38 K	6.6 s	2.5 s	0.3 s
BONE	54 K	87	15	17%	45 K	7.4 s	3.0 s	0.8 s
CAMILLE HAND	103 K	142	26	18%	74 K	12.5 s	5.6 s	1.2 s
CYLINDER	12 K	26	5	19%	14 K	2.4 s	0.9 s	0.1 s
SPHERE	19 K	7	2	29%	6 K	0.9 s	0.4 s	0.1 s
CUBE SPHERE	11 K	10	4	40%	4 K	0.6 s	0.3 s	0.0 s
TETRAHEDRON	14 K	4	2	50%	2 K	0.3 s	0.2 s	0.0 s
FANPART	5 K	5	3	60%	2 K	0.3 s	0.1 s	0.0 s
PRISMA	21 K	3	2	67%	3 K	0.5 s	0.4 s	0.0 s

Table 5.2: Statistics on a dataset of seamless parametrizations of tetrahedral meshes. Columns show the number of tetrahedra in the input meshes (*tets*), the number of triangular mesh facets tagged by algorithm 5.2 (*facets*), and the time spent in the three algorithmic steps (tracing the fire walls, building a T-mesh representation of the complex (including torus splitting), and wall retraction for reduction). Notice that, similar to table 5.1, the BC again has up to 30× as many blocks as the MC of the same model.

Algorithm 5.3: SERIAL MOTORCYCLE COMPLEX OF A HEX MESH

```
   foreach facet f on a critical surface do tag f
1  foreach edge e on a critical link or incident to a critical node do
       foreach facet f ∈ ∂⁻¹e do
2          if necessary(e, f) then Q.push((e, f, 0))                    // ignite
           while Q non-empty do
               (e, f, d) ← Q.pop()
               if uncrossed(e) and untagged(f) then                     // stop at burnt terrain
                   tag f                                                // burn
                   foreach uncrossed non-critical edge e′ ∈ ∂f do
                       if opp(e′, f) is not tagged then                 // propagate front
                           Q.push(e′, opp(e′, f), d + 1)
```

serial rather than simultaneous manner. While this voids canonicity[2], it enables the option to omit tracing motorcycles that would form removable (though not regular-removable) traces right away. For instance, if the two directions neighboring the next motorcycle's direction around a critical link have been traced (out of or into the critical link) already, the motorcycle can be omitted. The result (called *sparse MC* in the following) is coarser, though not necessarily irreducible.

Following this idea, algorithm 5.3 is a modified variant of the mesh-based algorithm 5.1; the parametrization-based algorithm 5.2 can be modified analogously. The key difference is that fire sources (facets incident at singularities) are processed one after the other, and those that are not necessary to establish a valid configuration around a critical link are skipped. In the loop (algorithm 5.3) we prioritize edges that already have some incident burnt facets, and process facets $f \in \partial e$ in circular order.

The condition necessary(e, f) (algorithm 5.3) is defined as follows. Let $f_{-2}, f_{-1}, f, f_{+1}, f_{+2}$ denote the (possibly cyclically self-overlapping) sequence of facets incident at critical edge e

[2]While the simultaneously constructed 2D motorcycle graph yields a *canonical* decomposition, its serial construction voids this property due to order dependence. For 3D neither algorithm yields a canonical partition, as already the crucial 2D right hand arbitration rule does not extend to 3D.

Model	raw	MC	MC_s	MC_{rs}	$\frac{MC}{MC_s}$	$\frac{MC}{MC_{rs}}$
EXAMPLE 3	9087	2877	5125	2691	56.1%	106.9%
EXAMPLE 1	3137	1123	1199	962	93.7%	116.7%
EXAMPLE 2	233	87	280	101	31.1%	86.1%
DRAGON-HEX	979	357	399	317	89.5%	112.6
GARGOYLE	720	257	283	220	90.8%	116.8%
ANC101 A1	1359	460	524	422	87.8%	109.0%
FERTILITY-HEX	221	76	80	66	95.0%	115.2%
PEGASUS-HEX	1035	374	408	287	91.7%	130.3%
KISS HEX	543	200	244	189	82.0%	105.8%
ANC101	609	207	136	120	152.2%	172.5%
IMPELLER STRESSTEST	184	37	71	61	52.1%	60.7%
ARMADILLO HEX-A	680	266	292	227	91.1%	117.2%
ARMADILLO HEX-B	396	147	176	133	83.5%	110.5%
			⋮			
EXAMPLE 5	1	1	1	1	100.0	100.0%

Table 5.3: Using the dataset from table 5.1, reported are the number of blocks in the raw motorcycle complex (raw), fully reduced motorcycle complex (MC), sparse serial motorcycle complex (MC_s) and its reduced version (MC_{rs})

Model	raw	MC	MC_s	MC_{rs}	$\frac{MC}{MC_s}$	$\frac{MC}{MC_{rs}}$
ARMADILLO	392	132	207	121	63.8%	109.1%
BONE	57	15	25	18	60.0%	83.3%
BROKEN BULLET	25	5	11	9	45.5%	55.6%
CAMILLE HAND	75	26	42	27	61.9%	96.3%
CUBE SPHERE	10	4	6	6	66.7%	66.7%
CYLINDER	11	5	5	5	100.0%	100.0%
FANDISK	43	19	17	14	111.8%	135.7%
FANPART	5	3	5	3	60.0%	100.0%
JOINT	54	15	21	14	71.4%	107.1%
KITTEN	77	19	53	30	35.8%	63.3%
PRISMA	3	2	2	2	100.0%	100.0%
ROCKERARM	217	78	183	114	42.6%	68.4%
SCULPTURE	27	13	18	13	72.2%	100.0%
SPHERE	7	2	5	2	40.0%	100.0%
TETRAHEDRON	4	2	3	2	66.7%	100.0%

Table 5.4: Using the dataset from table 5.2, reported are the number of blocks in the raw motorcycle complex (raw), fully reduced motorcycle complex (MC), sparse serial motorcycle complex (MC_s) and its reduced version (MC_{rs})

surrounding facet f (in either orientation). Condition necessary(e, f) is true iff f is not burnt yet and either:

- f_{-1} and f_{+1} are not burnt yet,
- f_{-1} and f_{+2} are burnt but not f_{+1}, or
- f_{-2} and f_{+1} are burnt but not f_{-1}.

In these cases f needs to be burnt (i.e., become part of the motorcycle complex as well) as well—otherwise the complex would contain cells with edges with inner angles of 270° (or larger) and would not be a pure cuboid block decomposition.

In essence, the algorithm attempts to omit "every other" (to the extent permitted by parity) wall around a critical link right away, rather than achieving this via reduction by wall retraction afterwards. Note that the incorporation of a similar omission strategy directly into the *non-serial* algorithm (as done for the 2D case in [Schertler et al. 2018, §3.1]) would not be straightforward because the a priori omission decision cannot be made simply per critical link in isolation but would require some form of global coordination in the interconnected network of singular arcs in the 3D case.

The result is a block decomposition that can be expected to be coarser than the immediate result (without reduction by wall retraction) of the algorithms from section 5.6. Indeed this is the case; however, our proposed *reduced* motorcycle complex typically is even simpler than this sparse serial motorcycle complex, as evident from tables 5.3 and 5.4. Of course reduction can also be applied to the sparse MC, but this yields no consistent benefit (last column). Note also that the sparse MC is incompatible with applications that admit only a regular-reduced MC.

Formalization as Embedded T-meshes

6

IV Quantize the seamless map ϕ into an integer-grid map ϕ_q.

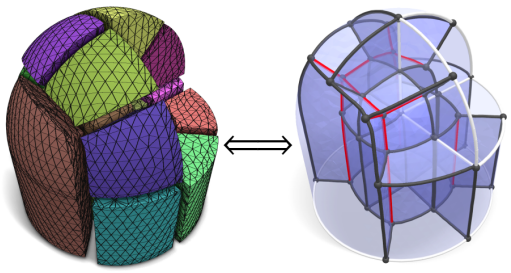
IVb Partition the volume into cuboids (w.r.t. ϕ)

… Obtain ϕ_q by quantizing the partition …

⇕

IVb Embed a ϕ-aligned T-mesh in the volume

… Obtain ϕ_q by quantizing the T-mesh …



In the last chapter we introduced the 3D motorcycle complex, the result of a volume decomposition algorithm on seamless parametrizations of volumes. This decomposition splits the input volume into a set of parametrically cuboidal blocks, each bounded by parametrically rectangular walls. Different from the base complex, however, the partition is non-conforming, hence, its blocks and walls can meet in T-junctions, allowing for a coarser decomposition. We furthermore outlined a description of its elements (blocks, patches, arcs, and nodes) and their connectivity as abstract cell complexes. In this chapter, we refine this description further, to properly capture the specific properties of these decompositions and establish a common notation for subsequent chapters. The formalism introduced here is not limited to decompositions obtained as motorcycle complexes, but applies to any non-conforming cuboid volume partition. In such a partition, each block has six pairwise opposite facets, geometrically speaking. However, connectivity-wise the facets of adjacent blocks do not match up one-to-one. In the following we reconcile this discrepancy via *T-meshes*, capturing both the mesh-like connectivity and the rectangular geometry of its cells. The concrete geometric realization of such T-meshes, being partitions of another volume in our case, is implied by how each of its cells is embedded into the discrete background (tetrahedral) mesh.

6.1 Related Work: Mesh-Embedded Meshes

Coarse meshes, either triangular, quadrilateral, or polygonal, embedded in a background triangulation (of a surface or in the plane), are used in a variety of scenarios. For instance, *quad layouts* are coarse quadrilateral meshes, typically embedded into a surface triangle mesh [Campen and Kobbelt 2014; Born et al. 2021]. Base complexes defining the parametric domain for spline surface representations are coarse (triangular or quadrilateral) meshes embedded in triangle meshes [Eck and Hoppe 1996]. In the context of texturing and similar mapping problems, *polycube partitions* are certain quadrilateral meshes embedded in surface triangle meshes [Tarini et al. 2004; Lin et al. 2008; Livesu et al. 2013], and *mesh triangulations* are coarse triangle meshes embedded in finer triangle meshes [Praun et al. 2001; Kraevoy et al. 2003]. Implementation-wise, intrinsic

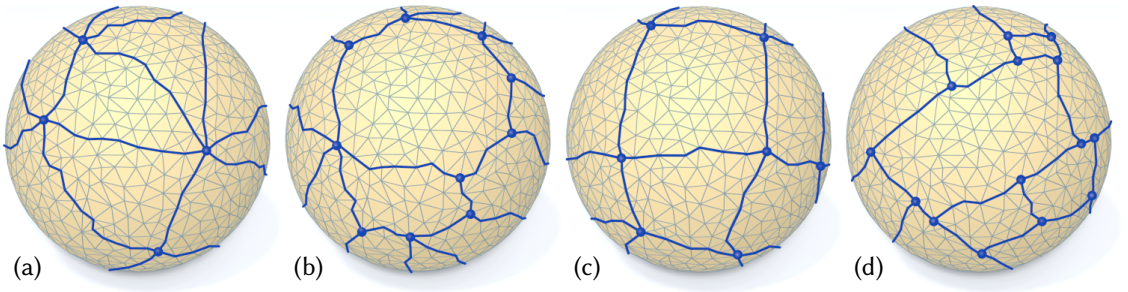


Figure 6.1: Illustration of 2D cell complexes embedded in a surface. (a) simplicial, (b) polygonal, (c) quadrilateral, conforming, (d) quadrilateral, nonconforming.

triangulations are often handled as embedded into a geometrically equivalent triangle mesh [Fisher et al. 2007; Sharp et al. 2019; Gillespie et al. 2021]. Also polygonal [Tong et al. 2006a; Born et al. 2021] and non-conforming meta meshes (with T-joints) [Lyon et al. 2019; Nuvoli et al. 2019; Lyon et al. 2021b; Pietroni et al. 2021] are made use of. The embedded mesh is sometimes referred to as *meta mesh*, whereas the mesh it is embedded into may be called *background mesh*. For representational simplicity, the embedding is often chosen such that a meta mesh edge (or *arc*) is embedded into a path of background mesh edges, and a meta mesh face (or *patch*) into a connected set of background mesh faces. See fig. 6.1 for an illustration. Volumetric meta meshes, embedded in a background tetrahedral mesh, appear more recently in particular in the context of mesh generation, in conforming [Takayama 2019; Livesu et al. 2020] varieties. Base complexes of hexahedral meshes [Gao et al. 2015a, 2017b] can be viewed as meta meshes embedded in a hexahedral background mesh.

6.2 Notation

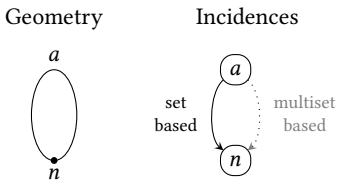
The key to formalizing non-conforming cuboid partitions as a mesh, as introduced in section 5.5.1, is to decouple connectivity and geometry: Splitting each side of a block (or rectangle) into smaller parts (blocks, arcs, patches and nodes), such that adjacent cells share only full parts, yields a mesh-like connectivity. The geometric interpretation of blocks (or patches) as cuboids (or rectangles), despite having more facets (or edges) than strictly necessary, is then captured by labeling which parts belong to which side. As this labeling prescribes the interpretation of angles between patches and blocks as either 90° or 180°, it specifically captures the T-shaped junctions of non-conforming partitions. In our case the labeling is induced by the underlying seamless parametrization, but once established, it can be processed detached from the parametrization and any other methods of obtaining a similarly-labelled partition can be used alternatively. The combination of mesh-like connectivity and side labelling, we call an *abstract T-mesh*. A T-mesh, theoretically, could refer to other (non-cuboid) non-conforming partitions as well. In our setting there is no ambiguity, though; the term as used here implies that blocks are cuboids. The concrete geometric realization of a T-mesh in our case is defined by its discrete embedding into the background tet mesh.

Notation 6.1 (n-cells of a T-mesh). To clearly differentiate the n-cells of polyhedral volume meshes (theorem 2.1) and those of T-mesh partitions, we use different names and symbols for the two. The cell complex representing a T-mesh is formed by the union of all its n-cells C_n with $n \in \{0, 1, 2, 3\}$: $C = C_3 \cup C_2 \cup C_1 \cup C_0$. We denote 3-cells as **b**locks $C_3 := B = \{b_1, b_2, \dots\}$, 2-cells as **p**atches $C_2 := P = \{p_1, p_2, \dots\}$, 1-cells as **a**rcs $C_1 := A = \{a_1, a_2, \dots\}$, and 0-cells as **n**odes $C_0 := N = \{n_1, n_2, \dots\}$.

6.2.1 Self-Adjacency

A first aspect to consider, is that the elements of coarse partitions (due to not being realized via simple linear shapes) can be self-adjacent (i.e., different parts of their boundary are glued together) without this implying a geometric defect. As described in the last chapter, this also commonly occurs in motorcycle complexes. While for these, it can be fully prevented by introducing additional walls (section 5.6.2.3) the quantization algorithms discussed in later chapters require the additional flexibility of self-adjacency being permitted. Hence, a slight generalization of the incidence notation from section 2.1.2 is required.

Consider the case of a cyclic arc a starting and ending at the same node n: ∂a, due to being set-based, returns only a single element, obscuring the fact that both ends of the arc connect to a node. We consider in the following an alternative where ∂ returns a multiset, and derive the induced bottom-up incidences ∂^{-1} and transitive incidences $\partial^{\pm k}$.



Notation 6.2 (Multiset logic). A multiset, different from a set, can contain the same element multiple times; the number of times is referred to as the *multiplicity* of an element. Let us denote a multiset X via an ordered pair of a set U (called a universe of elements) and a function $m_X : U \to \mathbb{Z}_0$ returning the multiplicity in the modeled multiset of any element in the universe. The multiplicity of an element can be zero; hence the domain of any multiplicity function m of a multiset that only draws elements from U is fixed. Basic set operations extend to multisets X, Y over the same universe U as follows:

Cardinality X has cardinality $|X| = \sum_{u \in U} m_X(u)$. We consider only multisets with finite cardinality.

Inclusion X is included in Y, denoted $X \subseteq Y$, iff $\forall u \in U : m_X(u) \leq m_Y(u)$.

Union The union Z of X and Y, denoted $Z = X \cup Y$, is obtained via $\forall u \in U : m_Z(u) = \max(m_X(u), m_Y(u))$.

Intersection The intersection Z of X and Y, denoted $Z = X \cap Y$, is obtained via $\forall u \in U : m_Z(u) = \min(m_X(u), m_Y(u))$.

Difference The difference Z of X and Y, denoted $Z = X \setminus Y$, is obtained via $\forall u \in U : m_Z(u) = \max(m_X(u) - m_Y(u), 0)$.

Sum The sum Z of X and Y, denoted $Z = X \uplus Y$, is obtained via $\forall u \in U : m_Z(u) = m_X(u) + m_Y(u)$.

The notation $x \in X$ indicates that $m_X(x) \geq 1$; when used as an index to denote iteration over all elements of X, iteration happens $m_X(x)$ times for each x.

In our case, U is handily identified with the entire T-mesh cell complex, i.e., $U := \mathcal{T} = B \cup P \cup A \cup N$. A multiset X over the T-mesh elements then is described sufficiently by just m_X.

Multiset incidences We alter the definition of ∂ so that it returns a *multiset* as defined above. Consider again the example of a loop edge; $\partial a = \{n, n\}$ is now valid, and explicitly models that both endpoints of an edge are the same node. Inversely, $\partial^{-1} n = \{a, a\}$ should also be true; after all, locally n sees two incident ends of an arc. This is achieved by inductively defining ∂^{-1} via $m_{\partial^{-1} c_1}(c_2) = m_{\partial c_2}(c_1)$. The remaining concepts, such as transitive bounding relations $\partial^{\pm k}$, closure $\langle C \rangle$ and star C^* retain their definition from section 2.1, accounting for the generalized (multi)-set operations listed above.

6.2.2 Labeling

Beside a more general connectivity, allowing for self-adjacency, T-meshes are characterized by a geometric interpretation of n-cells as parametrically aligned hyperrectangles: Blocks are cuboids bounded by six iso-planes, patches are rectangles bounded by four iso-lines, and arcs are iso-line segments. Modelling this does not require a full-blown parametrization. It is sufficient to only specify for each block and its bounding patches, which of the six sides of a cuboid each patch lies in, and, additionally, for each patch and its bounding arcs, which of the four edges of a rectangle each arc lies in. In case of self-adjacency, a patch or arc may occur on two sides. This information, together with the T-mesh connectivity, is sufficient to identify the 6 facets of a block (and all patches, arcs and nodes contained within), its 12 edges (and all arcs and nodes contained within), and its 8 corner nodes. The four edges and corners of a patch can likewise be deduced from this.

A valid hyperrectangle labeling $\mathcal{L}$ of blocks and patches (with 6 or 4 different pairwise opposite labels, respectively) can be formalized in different ways, but has to satisfy the following consistency conditions:

$\mathcal{L}_{\text{I}}$ Each $(n-1)$-dimensional bounding element is labeled (once by default, twice only in case of selfadjacency across that element).

$\mathcal{L}_{\text{II}}$ Each label is given at least once.

$\mathcal{L}_{\text{III}}$ Grouping bounding elements into maximal connected components (without crossing the bounds in case of self-adjacency) yields a number of components equal to the number of possible labels.

$\mathcal{L}_{\text{IV}}$ In the above grouping, the closures of groups with opposite labels are non-overlapping, i.e., sides labeled as opposite are not adjacent.

In our case, a valid labeling is immediately available from the parametrization; however, once this labeling is constructed, it can be processed detached from the parametrization, which comes in handy in chapter 9, which handles T-mesh modifications.

6.2.3 Abstract T-Mesh

Definition 6.3 (Abstract T-mesh). A cell complex $\mathcal{T} = B \cup P \cup A \cup N$ equipped with cell incidences ∂ and a hyperrectangle labeling $\mathcal{L}$ of blocks and patches is called an abstract T-mesh, if:

1. The bounds of each cell are also part of the mesh: $\forall s \in \mathcal{T}\, \forall s' \in \partial s : s' \in \mathcal{T}$.
2. Any two cells are either not adjacent or adjacent via one or more other full cells:

   $\forall c_1, c_2 \in \mathcal{T} : \langle c_1 \rangle \cap \langle c_2 \rangle = \varnothing \vee \exists C \subset \mathcal{T} : \langle c_1 \rangle \cap \langle c_2 \rangle = \langle C \rangle$
3. The mesh is a pure volume mesh:
   (a) Nodes are not bounded $\forall n \in N : \partial n = \varnothing$.
   (b) $\forall a \in A : |\partial a| = 2$, i.e., arcs are bounded by two nodes (not necessarily distinct ones).
   (c) $\forall p \in P : 2 \geq \left|\partial^{-1} p\right| \geq 1$, i.e., each patch bounds one or two blocks (not necessarily distinct ones).
   (d) $\forall a \in A : \left|\partial^{-1} A\right| \geq 1$, i.e., each arc bounds at least one patch.
   (e) $\forall n \in N : \left|\partial^{-1} n\right| 1$, i.e., each node bounds at least one arc.

4. $\mathcal{L}$ satisfies conditions $\mathcal{L}_{\mathrm{I}}$ to $\mathcal{L}_{\mathrm{IV}}$.

Technically, 2 is already implied by the definition of incidence and closure (cf. section 2.1.2), but is still listed here for completeness. Notably, the existence of a valid cube labeling implies that the cells of $\mathcal{T}$ are open n-balls. However, for self-adjacent cells the bounds of a cell do not form an n-sphere, hence their cell's closure closure is not a closed n-ball.

6.2.4 Embedded T-Mesh

Background Mesh We are working with 3D objects represented by tetrahedral meshes $\mathcal{M} = T \cup F \cup E \cup V$. For notational precision, we distinguish between an abstract element of $\mathcal{M}$ and its concrete geometric realization: For a vertex $v \in V$, we let $[v]$ denote the point in $\mathbb{R}^3$ it occupies. Similarly, for an edge $e \in E$, $[e]$ is an open line segment, for a facet $f \in F$ an open triangle, and for a tetrahedron $t \in T$ an open tetrahedron. Elements could be curved, by default we assume linear elements, though. The disjoint union $\bigcup_x [x]$ over all elements of $\mathcal{M}$ then is the compact set $[\mathcal{M}] \in \mathbb{R}^3$ occupied by $\mathcal{M}$. The tetrahedral mesh $\mathcal{M}$ is also referred to as the *background mesh* in the following; it serves as the basis to define the embedding of the T-mesh.

Notation 6.4 (Cell complex embedding). A discrete cell complex embedding of $\mathcal{T}$ into a tetrahedral mesh $\mathcal{M}$ is a map $\mathcal{I} : \mathcal{T} \hookrightarrow 2^{\mathcal{M}} \setminus \emptyset$, mapping each cell $c \in \mathcal{T}$ onto a *non-empty set* of elements of $\mathcal{M}$, respecting a number of embedding conditions. Here $2^{\mathcal{M}}$ denotes the set of all subsets of $\mathcal{M}$. Concretely, it can be viewed as the union of four maps, depending on the type of cell:

- $\mathcal{I}_0 : N \hookrightarrow V$
- $\mathcal{I}_1 : A \hookrightarrow 2^E \setminus \emptyset$
- $\mathcal{I}_2 : P \hookrightarrow 2^F \setminus \emptyset$
- $\mathcal{I}_3 : B \hookrightarrow 2^T \setminus \emptyset$

This means each node is mapped to a vertex, each arc to a set of edges, each patch to a set of facets, and each block to a set of tetrahedra. While this representation is convenient for implementation purposes, formal properties are more easily expressed using a derived map $\hat{\mathcal{I}}$ that additionally includes all interior lower-dimensional elements in a meta-cell's image: $\hat{\mathcal{I}}(c) \coloneqq \langle \mathcal{I}(c) \rangle \setminus \langle \mathcal{I}(\partial c) \rangle$. Application of $\hat{\mathcal{I}}$ to a set of cells $C \in \mathcal{T}$ is defined as $\hat{\mathcal{I}}(C) \coloneqq \bigcup_{c \in C} \hat{\mathcal{I}}(c)$. This also directly defines a meta-cell's geometry, via $[c] \coloneqq [\hat{\mathcal{I}}(c)]$, i.e., the geometric realization of cell c is defined by that of the elements it is embedded in. This is illustrated in fig. 6.2.

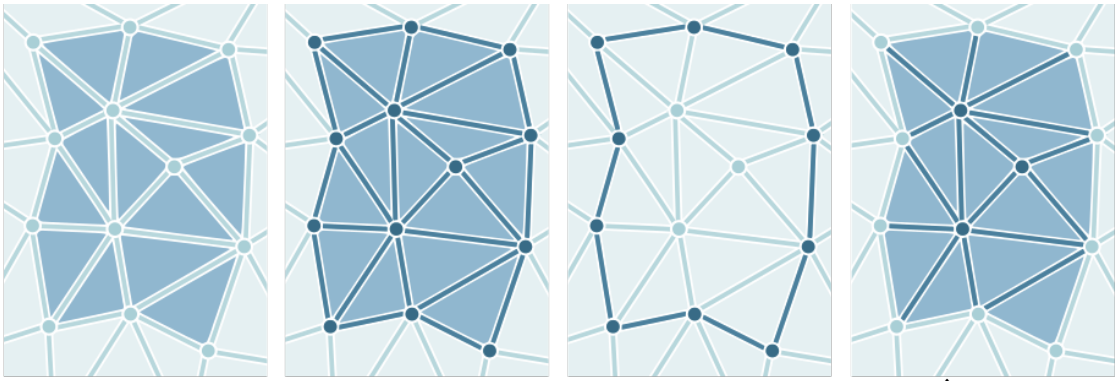


Figure 6.2: Illustration of a discrete embedding $\mathcal{I}(p)$ of a quadrilateral patch p (left) and its implied geometric realization $[p] = [\hat{\mathcal{I}}(p)]$ (right), topologically an open disk. In the center two intermediate definitions are illustrated.

Definition 6.5 (Embedded T-mesh). An abstract T-mesh $\mathcal{T}$ equipped with an embedding map $\mathcal{I}$ onto $\mathcal{M}$ is an embedded T-mesh iff:

($\mathcal{I}_{\text{I}}$) $\mathcal{I}$ is injective: $\hat{\mathcal{I}}(c_i) \cap \hat{\mathcal{I}}(c_j) = \emptyset$ for all $c_i \neq c_j \in \mathcal{T}$.

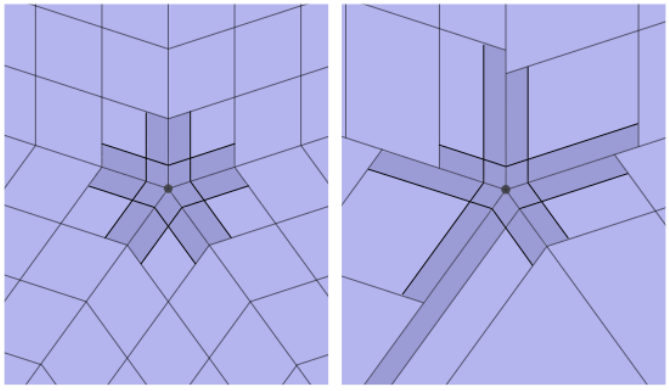
($\mathcal{I}_{\text{II}}$) $\mathcal{I}$ is surjective: $\hat{\mathcal{I}}(\mathcal{T}) = \mathcal{M}$.

($\mathcal{I}_{\text{III}}$) $\mathcal{I}$ is structure-preserving $\hat{\mathcal{I}}(\langle c \rangle \setminus c) = \left\langle \hat{\mathcal{I}}(c) \right\rangle \setminus \hat{\mathcal{I}}(c)$.

In some other use cases of cell complex embeddings, especially when the dimension of the meta-mesh is different from that of the background mesh, ($\mathcal{I}_{\text{II}}$) is relaxed. An example would be the embedding of the singular graph formed by singular nodes and singular links into a tetrahedral or hexahedral domain. Intuitively, ($\mathcal{I}_{\text{III}}$) holds if for each cell the 'image of its boundary' equals the 'boundary of its image'. An important consequence of structure-preservation is, that because the meta-cells of $\mathcal{T}$ are open n-balls they are mapped to subsets of $\mathcal{M}$ that are also open n-balls; both topologically and geometrically under $[c]$. If parts of the boundary of a meta-cell are glued together, resulting in self-adjacency, the same is true for its multi-cell embedding. In the motorcycle complex, discussed in the previous chapter, all the above aspects are guaranteed by construction.

6.3 Alternative Use Case: Solid T-Splines

T-splines are a flexible tool in the context of smooth function representation, for geometric modelling as well as for isogeometric analysis. For the volumetric case, constructions of T-spline spaces starting from hexahedral meshes have been described [Wang et al. 2012]. As solid T-splines are defined over cuboid complexes which are not necessarily conforming (hence the 'T'), it is actually unnecessarily restrictive to start from a conforming one, i.e., a hexahedral mesh. We can essentially apply the necessary structural refinement around singularities that the above paper describes directly on the T-mesh obtained via the motorcycle complex. This circumvents the need for quantization (to obtain a hex mesh), and effectively provides a coarser starting configuration—which could then be adaptively refined where necessary for a particular application, as opposed



Model	+BC	+MC	factor
1	12743	4190	3.0×
2	28352	14656	1.9×
3	14520	10736	1.4×
4	31217	15407	2.0×
5	17866	10608	1.7×
6	26785	14167	1.9×
7	14520	7348	2.0×
8	11165	6437	1.7×
9	24685	12720	1.9×
10	19030	11303	1.7×

Figure 6.3: T-spline-required refinement of blocks around a singular arc (cross section view) in a conforming complex (left, e.g. base complex) versus a non-conforming complex (center, e.g., motorcycle complex). Right: The T-mesh derived in this way from the motorcycle complex (MC) is typically much coarser compared to the base complex (BC); on 10 example models from table 5.1, the number of additional refinement-induced cells (+BC and +MC, respectively) is 1.4-3.0 times larger for the (already initially larger) BC.

to starting with a rather dense hexahedral mesh as domain structure and later coarsening it where possible.

In fig. 6.3 we illustrate the refinement around a singularity (inserting additional walls) by the rules of [Zhang et al. 2012], so as to yield a T-mesh suitable as control mesh for a solid T-spline. We note that some additional modifications to the control mesh structure can be necessary to ensure continuity due to non-local overlaps of basis function supports with singularities, as discussed in [Campen and Zorin 2017, §8.2], or to yield specific classes of splines, such as analysis-suitable splines, as discussed in [Scott et al. 2012]. In any case, the motorcycle complex T-mesh provides a significantly simpler starting point than the base complex (or a hexahedral mesh derived from it). We contrast the numbers of additional T-mesh blocks due to refinement-at-singularities applied to the BC and the MC in fig. 6.3.

6.4 Main Use Case: Quantization for Hex Meshing

A seamless parametrization (theorem 3.2) is the continuous relaxation of an integer-grid parametrization (theorem 3.1). In the 2D case, this relation is exploited in state-of-the-art quadrilateral mesh generation methods, discussed in section 3.5.3. Tong et al. [2006b] introduced the concept of parametrization seamlessness (only naming it differently), including a first rudimentary form of integer quantization applied after establishing seamlessness. Later, Kälberer et al. [2007] pioneered the idea of first generating a continuous seamless parametrization, and then *rounding* it (at once or iteratively [Bommes et al. 2009]) to a quantized seamless parametrization. This rounding is a notoriously fragile process, though: With increasing target quad size, the risk of yielding an invalid parametrization (with degenerate or flipped parts) increases, as pointed out and demonstrated in [Bommes et al. 2013, Fig. 1] and [Campen et al. 2015, Fig. 3]. An analogous rounding procedure has been described for the 3D case [Nieser et al. 2011]; it is the state-of-the-art approach to yield quantized volumetric seamless parametrizations, as evidenced by its sustained use in recent works [Fang et al. 2016; Solomon et al. 2017; Liu et al. 2018; Corman and Crane 2019; Palmer et al. 2020]. Not surprisingly it comes with the same limitations as its 2D counterpart, as also evidenced in table 6.1 and fig. 6.4.

In the 2D case, subsequent work has provided a remedy, taking a different path, reliable and efficient, from seamless to integer-grid parametrizations: Via 2D T-meshes [Campen et al. 2015; Lyon et al. 2019, 2021b]; for the 3D case, this path has not been paved yet. 3D T-meshes, as constructed via the motorcycle complex routine, are the key to extending this state-of-the-art approach to the 3D case, generating hexahedral meshes via volumetric seamless parametrizations.

6.4.1 Proof-of-Concept Quantization

Embedded T-meshes as proxy domains for quantization Recall that the main motivation behind the effort of constructing the motorcycle complex is, that trying to implement a robust quantization algorithm directly on the tetrahedral mesh—and the seamless map defined on it—is prone to run into efficiency issues due to the high complexity of the tetrahedral mesh, as was already shown for the 2D case (cf. section 3.5.3). Instead, the motorcycle complex offers a coarse decomposition of the domain, while still sharing the same topology as well as all critical entities (boundaries, singularities, features) with the original domain. Essentially, it represents a coarser proxy domain that still captures all relevant aspects, simply omitting details in "uninteresting" regions, i.e., those regions that do not require special attention in terms of map quantization. By formalizing it as a T-mesh, we can now treat it as a first-class domain for quantization, mostly detached from the background, while at the same time using mesh-based tools similar to ones we

would otherwise employ on the background mesh. In the following we devise and test a simple (but not particularly useful) quantization method, showcasing the potential of this approach, which is further elaborated in the subsequent chapters.

Approach outline We compute the motorcycle complex of a continuous seamless parametrization, and then scale the parametrization within each block to positive integer extents, so that each block becomes a parametric cuboid with integer dimensions $l \times m \times n$. Assuming any corner per block to be fixed on an integer-grid point, the resulting map obviously is an integer-grid map. Inversely, this prescribes a subdivision per block into $l \times m \times n$ hexahedra. The subdivision integers (the integer parametric extents per block) need to be chosen such that the subdivisions conform across block boundaries. This is achieved by expressing the block dimensions via integer length assignments for each arc—shared between patches and blocks, inherently ensuring compatibility. We call this the *T-mesh quantization*.

What we need to require for this assignment is that patches remain rectangles (thus blocks remain rectangular cuboids) parametrically. Let $A_i(p)$, $i \in \{0,1,2,3\}$, denote the set of arcs $a \in \partial p$ labeled as being on side i of patch p, and $q_a \in \mathbb{Z}^{>0}$ the length assignment of arc a. Then this requirement can be expressed using two linear constraints per patch:

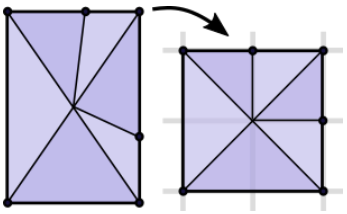
$$\forall p \in P \forall i \in \{0,1\} : \sum_{a \in A_i(p)} q_a = \sum_{a \in A_{i+2}(p)} q_a \tag{6.1}$$

One aims to reproduce the sizing of the given parametrization using

$$\min_q \sum_a (q_a - \ell_a)^2, \quad \text{s.t.} eq.\ (6.1) \tag{6.2}$$

where ℓ_a denotes the *target length* of arc a, sensibly chosen as its length under the seamless parametrization and potentially scaled via a global factor s to control the resulting mesh's resolution.

Map rescaling Note that the seamless map per block cannot be scaled by a simple affine map as each of the block's six facets consists of possibly multiple walls (due to T-joints from outside the block), and each wall needs to be scaled according to its arcs' values q_a. It can be achieved via a *piecewise*-affine map σ, though: Affine per tetrahedron spanned by the block's center point with a triangle in a conforming triangulation of the (rectangular) walls on its surface. The inset illustrates a 2D version. If one splits the underlying tetrahedral mesh by these meta-tetrahedras' faces, σ is affine per element and no inversions occur under $\sigma \circ \phi$ (most of this refinement is superfluous and can be omitted). A smoothing of either the resulting parametrization [Rabinovich et al. 2017] or the implied hex mesh [Livesu et al. 2015] can be applied subsequently to distribute distortion evenly.



Comparison to Rounding To give an idea of the benefit, in table 6.1 we compare this motorcycle complex based quantization strategy with the classical rounding strategy on a dataset of 19 tetrahedral meshes with frame fields, namely those shown in [Liu et al. 2018]. It can be seen in table 6.1 that for 15 of these a valid continuous seamless parametrization can be obtained to begin with—namely those used for the above experiments in table 5.2. To these 15 continuous parametrizations we applied the rounding strategy, increasing the target edge length (via scaling the seamless map ϕ by a global factor s) until failure, and state the number of hexahedra implied by the coarsest rounded seamless parametrization that could validly be obtained. We also applied

the motorcycle complex based quantization strategy, and state the number of hexahedra obtained when aiming for maximal coarseness (via $\ell_a = 0$ for all a). It can be observed that, except for the simplest models, typically a significantly coarser quantization, thus coarser hex mesh can be obtained. The key here is that using the T-mesh arcs as proxies to express integer extents of the underlying parametrization, yields a differential expression of the integer coordinates of critical entities; arc lengths prescribe parametric differences rather than absolute values. Hence, our simple proof-of-concept strategy, that employs $q_a \geq 1$ everywhere, trivially separates different critical entities by at least a parametric distance of 1, and thus prevents them from coinciding on the same integer-grid point, which would degenerate the map in between. The rounding approach operates on absolute, local parameters and can not make decisions globally, so it often rounds parametric coordinates of critical entities onto the same integer-grid coordinate—increasingly so for coarser targets.

Post-processing repair We remark that HexEx [Lyon et al. 2016] can sometimes extract valid hex meshes even from invalid rounded parametrizations. We applied it to parametrizations rounded to the same level of coarseness as can be achieved with our approach. In 7 of the 16 cases where the rounded parametrization is invalid, it was able to output a valid hex mesh; in 9 cases the hex mesh has defects, some examples of which are shown in fig. 6.4. This shows, that most likely for the former 7 cases the obstacle causing non-injective maps is not invalid integers obtained from rounding (these would not admit a valid hex mesh) but rather the general difficulty of

Table 6.1: Statistics on the maximum coarseness (number of hexes) of hexahedral meshes generated from seamless parameterizations using our approach employing the motorcycle complex T-mesh (MC), and via the classical method of iterative rounding (Round). For the bottom four models our method is inapplicable, as the initial seamless map contains inversions. We also report the percentage of parametrically inverted (or degenerate) tetrahedra in the output integer-grid map when trying to achieve the coarseness of MC (or a very fine mesh in the four bottom rows) with the rounding-based approach. As can be seen in the last column, the HexEx approach from [Lyon et al. 2016] is able to recover a valid hex mesh from these invalidly rounded parametrizations only in mild cases. Also see fig. 6.4.

Model	**MC**	$\frac{\textbf{MC}}{\textbf{Round}}$	**Round**	inverted	**HexEx**
KITTEN	176	5.7\%	3112	9.1%	broken
ARMADILLO	1884	6.2\%	30456	7.3%	broken
CAMILLE HAND	122	8.5\%	1438	5.6%	broken
BONE	87	13.9\%	628	4.8%	broken
SCULPTURE	108	16.8\%	642	1%	valid
SPHERE	7	21.9\%	32	5.3%	valid
FANDISK	110	21.9\%	502	1.3%	valid
JOINT	205	23.1\%	888	1.6%	broken
ROCKERARM	1391	29.1\%	4784	2.7%	broken
PRISMA	3	33.3\%	9	2.3%	valid
FANPART	5	38.5\%	13	0.6%	valid
CYLINDER	26	43.3\%	60	3.6%	valid
CUBE SPHERE	10	100.0\%	10	0%	valid
TETRAHEDRON	4	100.0\%	4	0%	valid
BROKEN BULLET	44	122.2\%	36	0%	valid
STAR BOLT	-	-	-	0.1%	valid
KNOT	-	-	-	0.9%	broken
BUNNY	-	-	-	0.1%	broken
ELEPHANT	-	-	-	0.2%	broken

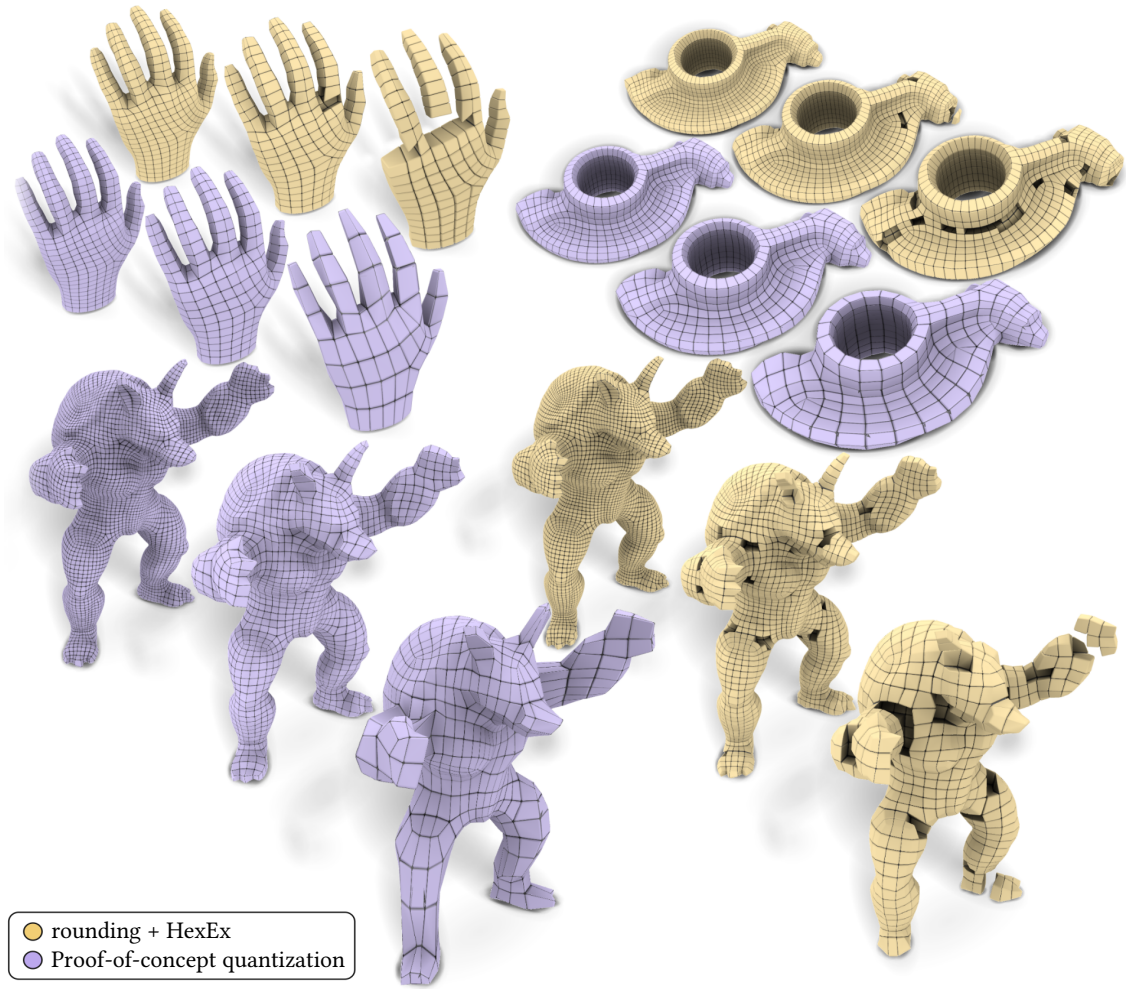


Figure 6.4: Hexahedral meshes of increasing coarseness obtained from seamlessly parametrized tetrahedral meshes by means of iterative rounding to an integer-grid map as described by [Nieser et al. 2011] (dark gray) and our proof-of-concept quantization using the motorcycle complex (white). When increasing target coarseness, hexahedral meshes obtained by rounding become increasingly defective due to parametric degenerations and inversions. Even fault-tolerant mesh extraction [Lyon et al. 2016], as employed here, cannot recover from this, leaving gaps that cannot easily be patched.

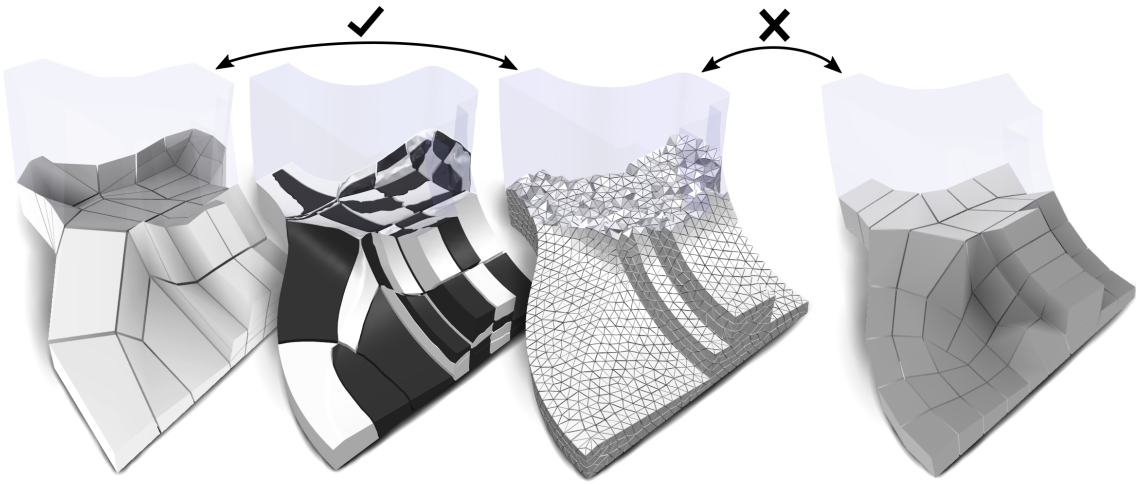


Figure 6.5: Left: The quantization approach yields a hex mesh (left) with volumetric correspondence between each hex and a region (checkerboard piece) of the input object. Right: A hex mesh recovered by HexEx does not come with such a volumetric map.

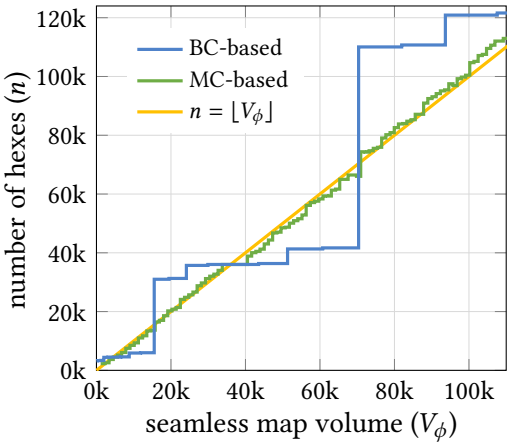


Figure 6.6: Mesh resolution can be controlled more finely when using the MC, not the BC, as basis for quantization. This is illustrated here for model EXAMPLE_2 (third model from table 5.1). The number n of hexahedra in the extracted hex mesh is shown versus the parametric volume V_ϕ, which captures the *targeted* number of hexes: Ideally, $n = \lfloor V_\phi \rfloor$.

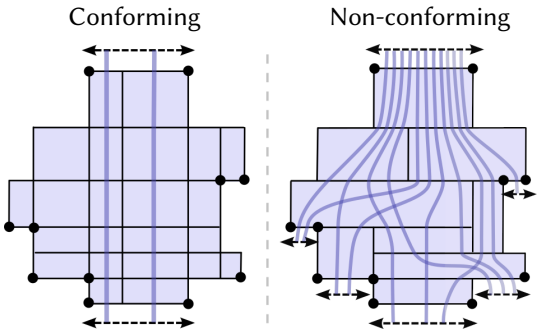


Figure 6.7: Illustration of a 2D slice through a conforming (left) or non-conforming T-mesh (right); black dots are selected critical points. Assume the number of sheets of hexahedra passing between the upper two singularities shall be changed. In the conforming partition, this can be achieved using one of two *sheet operators* (purple paths), with identical effect. In the T-mesh, there are 11 different sheets, with a choice of different effects on other critical points.

volume parametrization between non-trivial domains; this, too is side-stepped by the block-wise parametrization rescaling. Note also that when HexEx succeeds in ignoring the parametrization's defects, it does output a mesh but, in contrast to our approach, no valid parametrization, in particular no bijection between the input and the output mesh (fig. 6.5).

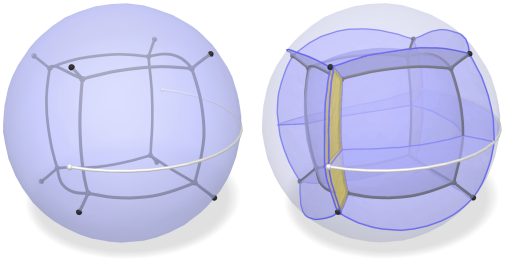
Alternative: Coarsening. One may consider the alternative of only applying mild (therefore more robust) rounding, yielding an overly fine initial hex mesh, followed by coarsening, e.g., using [Gao et al. 2017b]. Obvious downsides are the lack of a priori knowledge of a successful target edge length setting (potentially requiring trial-and-error) as well as the higher time and memory cost of this approach, operating fine-to-coarse rather than coarse-to-fine. Furthermore, the conforming sheet operators employed for structure and singularity preserving mesh coarsening are more restricted than a motorcycle complex based quantization procedure, cf. fig. 6.7.

Alternative: Base Complex. For the same reason (in addition to complexity-related reasons), it is better to formulate the quantization problem eqs. (6.1) and (6.2) based on the coarser and non-conforming motorcycle complex than on the base complex: There are more degrees of freedom, more ways for the solver to adjust the quantization length assignment q (while respecting eq. (6.1)), as illustrated in fig. 6.7. This in particular enables fine-grained control over the resulting mesh sizing (fig. 6.4). Figure 6.6 illustrates that when basing the quantization system on the base complex rather than the motorcycle complex, density control may be significantly less fine-grained. This is intimately related to the lower number of degrees of freedom in the conforming structure of the base complex (cf. fig. 6.7).

6.4.2 Limitations

The here outlined naive quantization approach is simply meant to demonstrate advantages over partition-less strategies like fragile rounding or hypothetical strategies based on conforming partitions. In the current form it is far from practicable in a general setting; this is because requiring positive-only quantizations ($q_a \geq 1$) forcibly prevents parametric alignment of critical entities

in the integer-grid map—unless they are *exactly* aligned already in the continuous seamless map, which is extremely unlikely. This forced mis-alignment of entities results in a much more complex block structure of generated hex meshes (as discussed in section 2.3.3.2), and is especially detrimental for pairs of entities that are *almost* exactly aligned. For these, tiny parametric differences in the seamless map are blown up to entire hexahedral sheets in the output, causing not only structural deterioration but also geometric distortion of elements. In the inset, such a misalignment of singularities (black) and features (white) is demonstrated, manifesting itself as a pathologically slim block (orange). Requiring $q_a \geq 1$ forces this block to be filled by a layer of hexahedra; instead, a parametric width of 0 for that block is desirable but precludes the simple block-wise parametrization rescaling.



Related work on layout subdivision The concept of quantization, fixing exactly the integer degrees of freedom that determine the number of quads or hexahedra to pave each region of a domain with, has been the focus of sustained research also in the engineering and meshing community. In [Tam and Armstrong 1993] an integer linear programming method was developed, assigning subdivision numbers to conforming patch and block layouts while minimizing the absolute deviation of those numbers from non-integer targets. The often prohibitively slow branch-and-bound integerization, as well as the inability of the objective to prevent error concentration in just a few variables, were improved on in a series of later works, minimizing the deviations lexicographically [Mitchell 2000], and opting for heuristic [Mitchell 2014] and incremental [Mitchell 2023] approaches instead. In contrast to our work, all the aforementioned methods deal with conforming, tidy layouts of rather uniformly sized patches or blocks. As such, they do not need to incorporate the possibility of removing layout elements within the quantization—achieved by allowing 0 as a valid quantization value—and therefore preserve the critical structure of the domain "for free," just like the proof-of-concept quantization outlined above.

Full-fledged quantization In the following chapters, we derive a full-fledged quantization strategy based on T-meshes that overcomes these limitations, by allowing quantization values of 0, while retaining the following guarantees:

- Validity of the chosen integers, in the sense that they always admit an injective integer-grid map (chapter 7),
- Fixation of only exactly the integer variables of the global integer-grid map, leaving the continuous variables free for distortion optimization (chapter 8), and
- Injectivity of the final integer-grid map (chapter 9).

Part C

Robust Volume Quantization

heu·ris·tic /hjuːˈrɪs-tɪk/ *n.*
An algorithm that doesn't work.

Jeff Erickson

7 T-mesh Quantization

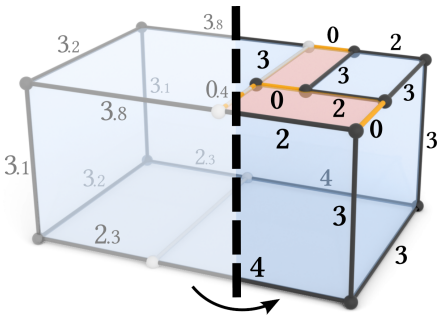
IV Quantize the seamless map ϕ into an integer-grid map ϕ_q.

IVa–b Embed a ϕ-aligned T-mesh $\mathcal{T}$ in the volume

IVc Quantize $\mathcal{T}$ into $\mathcal{T}_q$

… Obtain ϕ_q from $\mathcal{T}_q$ …

V Extract a hex mesh from ϕ_q.



This chapter builds on the naive quantization routine from section 6.4.1, to address the following question: How to assign integer lengths to T-mesh arcs, such that this

(a) implies a structure-preserving integer-grid map of the background mesh with the same topology as the input seamless map,

(b) restricts the solution space of integer-grid maps as little as possible,

(c) achieves a faithful approximation of continuous target sizes.

Concretely, aspect (a) requires the formulation of quantization constraints ensuring pairwise separation of critical entities (singularities, features, and boundaries) and preservation of the domain topology. Aspect (b) demands that such constraints should be as non-restrictive as possible while being sufficient. Lastly, aspect (c) asks for a sensible quantization objective, for which an optimization procedure retrieves quantized solutions within the so demarcated feasible set, that deviate from the intended target as little as possible. In the following, after formalizing these requirements, we devise a concrete formulation of the general T-mesh quantization problem as an *integer quadratic program* (IQP), and afterwards develop efficient solution strategies for this specific class of IQP instances.

7.1 Notation

Notation 7.1 (T-mesh quantization). Let $\mathcal{T} = B \cup P \cup A \cup N$ be an abstract T-mesh as per theorem 6.3, consisting of blocks, patches, arcs and nodes. A T-mesh quantization is represented as an integer vector $q = (q_1, \ldots, q_{|A|})$, assigning an integer q_i to each arc $a_i \in A$.

For each arc a_i the quantized length q_i denotes its parametric length under the to-be-computed integer-grid map, and—roughly speaking—implies the number of hexahedral sheets to be placed along that arc in the later hexahedral mesh. The total lengths of arcs on opposite sides of the same patch must match to guarantee the existence of a corresponding quadrilateral subdivision per patch, and thus a conforming hexahedral subdivision per block and for the entire domain.

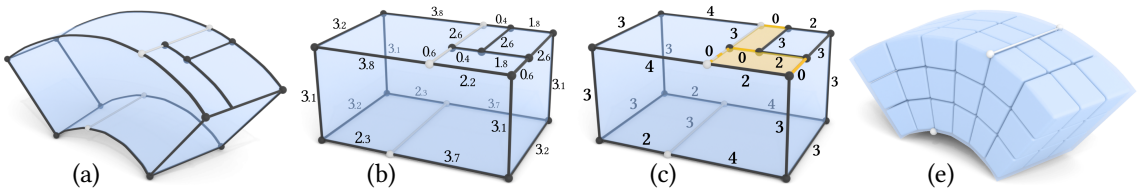


Figure 7.1: An illustration of the principle behind our T-mesh quantization: (a) A T-mesh is constructed on the input domain, possibly with features (white). (b) Under the input seamless map, each T-mesh element has a non-integer size measurable via continuous T-mesh arc lengths. (c) T-mesh lengths are consistently quantized to integers, including 0 (orange). (d) The resulting quantized T-mesh implies a certain number of hexahedral sheets crossing each arc, patch, and block.

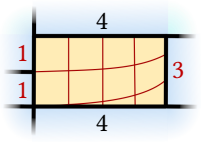
Notation 7.2 (Quantization targets). The T-mesh quantization is required to faithfully approximate target continuous lengths, represented as a real-valued vector $\ell = (\ell_1, \ldots, \ell_{|A|})$, assigning a target length ℓ_i to each arc a_i.

In our concrete case, T-meshes are constructed on seamless parametrizations ϕ, which are continuous relaxations of integer-grid maps (section 3.2). Here, a sensible choice is to assume optimal target lengths per arc are exactly the parametric lengths of arcs under ϕ. By virtue, achieving a close approximation of target arc lengths indirectly also preserves patch and block extents from the seamless map in the implied integer-grid map. Hence, the quantization objective is to minimize some distance $d(q, \ell)$. We say, that a quantization q is more *accurate* than another one q', iff $d(q, \ell) < d(q', \ell)$.

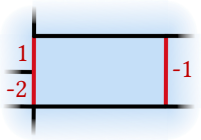
As pathologically slim regions, discernable by $\ell_i < 0.5$, commonly occur in the T-mesh (see section 6.4.1), being able to locally assign $q_i = 0$ is crucial for minimizing $d(q, \ell)$ and avoiding large distortion in the integer-grid map. Permitting zeros, however, introduces a risk: All critical entities must remain represented and mutually separated in the later hexahedral mesh; hence, some regions need to be constrained to greater-than-zero size. Furthermore, while we show that handling individual arc with negative integer extents (implying a parametric direction inversion), as well as handling entire blocks with a quantized extent of 0 is feasible, allowing for negative block extents (i.e., parametrically inverted blocks) must be prevented for quantizations to remain practicable. In summary, this leads to the following definition of quantization *feasibility* (or equivalently, *validity*).

Definition 7.3 (Quantization feasibility). A T-mesh quantization q is feasible *iff* it is:

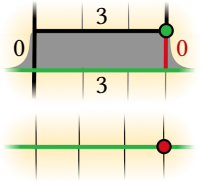
Consistent, meaning that length sums of arcs on opposite patch sides must match (eq. (6.1)). This intuitively corresponds to equally many quad strips (or hex sheets) passing into and out of the patch, and can be modeled via a system of equality constraints $\mathcal{A}q = \mathbf{0}$. Inconsistencies like in the inset are avoided this way.



Non-negative, meaning that blocks must not have negative extents along any axis, like shown in the inset, as negative block sizes would imply volume inversion. This can be modeled with inequality constraints of the form $\mathcal{B}q \geq \mathbf{0}$.



Structure-preserving, meaning that the topology of the domain and its critical entities (singularities, features, and boundaries) must not change under the metric implied by q, as this would result in degeneration of the quantized map or in loss of features (green), as exemplified in the inset.



Structure-preservation essentially demands that the size of critical entities as well as spacings in between must remain greater-than-zero. This can be modeled via inequality constraints $Cq \geq \mathbf{1}$. We show in section 7.3.4 that ensuring this, due to the coupling of the singular structure to the background domain, also ensures topology preservation for the entire background domain. Note specifically, that the assignment of $q = \mathbf{0}$ is consistent and non-negative, but not structure-preserving, hence not feasible. The solution of naive rounding approaches often yielding defects is also due to the integer assignment not being structure-preserving. In the following, the constraints and the objective of T-mesh quantization is explored in further detail.

7.2 Formulation as an Integer Program

Formally, the T-mesh quantization problem detailed above can be modeled as a linearly constrained integer problem:

$$\begin{aligned} \min \quad & d(q, \ell) && \text{(7.1a)} \\ \text{s.t.} \quad & \mathcal{A}q = \mathbf{0} \quad \text{(consistency)} && \text{(7.1b)} \\ & \mathcal{B}q \geq \mathbf{0} \quad \text{(non-negativity).} && \text{(7.1c)} \\ & Cq \geq \mathbf{1} \quad \text{(structure preservation)} && \text{(7.1d)} \end{aligned}$$

Quantization feasibility as defined in theorem 7.3 translates directly over to the fulfilment of the constraints eqs. (7.1b) to (7.1d). This problem can be solved by employing a general-purpose black-box integer solver, as is the default assumption in this chapter. In chapter 11 a special-purpose 'transparent-box' solver, tailored to the problem at hand, is developed and its advantages over general-purpose integer solvers demonstrated. In our concrete formulation here, we employ a quadratic objective function; in this case the above problem is an *integer quadratic program*, or *IQP* for short.

7.2.1 Objective Function

In the following we assume the objective function is:

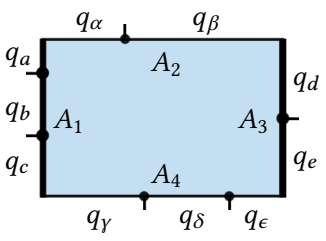
$$d(q, \ell) = \sum_{a \in A} (q_a - \ell_a)^2 \quad , \tag{7.2}$$

where ℓ_a is the parametric length of arc a under the seamless map ϕ. Essentially, it measures the length distortion of arcs between the input seamless map and a per-block integer-grid map of the same topology. An objective function that aims to instead reduce volume distortion, and operates on a global, rather than block-wise scale is discussed in chapter 10. The quadratic objective has the advantage of lending itself nicely to efficient convex optimization and also punishes large, isolated length distortions more harshly which is desirable to achieve a smooth resolution profile in the output.

7.2.2 Consistency

Let $A_i(p)$, $i \in \{1, 2, 3, 4\}$, denote the set of arcs $a \in \partial p$ labeled as being on side i of patch p. A quantization is *consistent* iff

$$\forall p \in P \forall i \in \{1, 2\} : \quad \sum_{a \in A_i(p)} q_a = \sum_{a \in A_{i+2}(p)} q_a \quad . \tag{7.3}$$



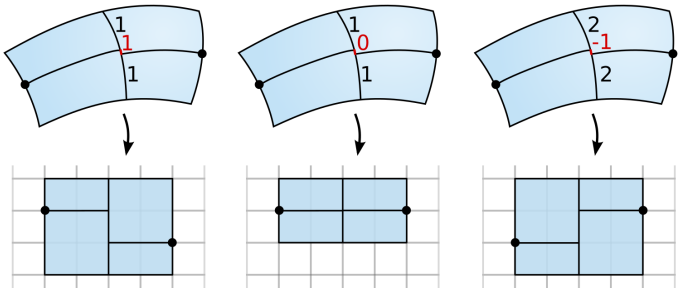


Figure 7.2: 2D illustration of the effect of $\mathbb{Z}^{>0}$ vs $\mathbb{Z}^{\geq 0}$ vs $\mathbb{Z}$. Two singularities (dots) are slightly offset in the input parametrization (top). For $q_a \in \mathbb{Z}^{>0}$, this misalignment is even amplified to at least 1 (left). Allowing $q_a \in \mathbb{Z}^{\geq 0}$ (center), the singularities can be aligned. Allowing negative values, $q_a \in \mathbb{Z}$ (right), even their relative order may be changed, if globally beneficial.

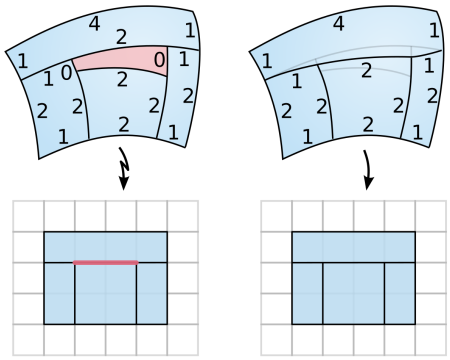
This means each patch is rectangular under quantization q. Note that for target lengths ℓ derived from an underlying seamless parametrization this property holds, but not necessarily for quantized arc lengths obtained from rounding each such target length, as evident from the inset, if for instance $\ell_a = \ell_b = \ell_c = 0.33$ and $\ell_d = \ell_e = 0.5$.

7.2.3 Non-Negativity Constraints

Let $A_i(b)$, $i \in \{1, 2, 3\}$, denote the set of arcs $a \in \partial^2 b$ for which the labeling $\mathcal{L}$ implies they lie on pairwise orthogonal block edges. A quantization is *consistent* iff We require that blocks have non-negative extent, i.e.

$$\forall b \in B \, \forall i \in \{1, 2, 3\} : \quad \sum_{a \in A_i(b)} q_a \geq 0 \quad . \tag{7.4}$$

Note that this does not require each individual arc to have a non-negative value. The additional degrees of freedom, offered by not restricting q_a to $\mathbb{Z}^{\geq 0}$ or even $\mathbb{Z}^{>0}$, are demonstrated in fig. 7.2.



Let us first discuss how a block with an assigned extent of zero (in one or multiple dimensions) fits the per-block rescaling intuition introduced in section 6.4.1. Of course, mapping a zero-block $b \in \mathcal{M}$ embedded via $\mathcal{I}(b)$ in a background mesh $\mathcal{M}$ onto a parameter cuboid of zero volume would imply a degenerate map for all contained tet mesh elements $c \in \mathcal{I}(b)$ (see the inset). We can, however, view the assignments in a more global scope. When considering the embedding of any zero-extent meta-cell of the T-mesh into $\mathcal{I}(b)$ to become collapsed, thereby distributing the meta-cell's 'content' to neighboring nonzero-cells that expand accordingly, map injectivity is maintained. Mapping the resulting 'empty' zero-extent meta-cells onto degenerate parameter cuboids then is perfectly compatible with an injective map. In this sense, zero-cells do not present a problem—unless they span a topologically incontractible cycle (a solid handle for instance) or their contraction involves endpoints whose embedding should remain fixed (critical entities).

In other words, the hypothetical contraction of zero-extent cells can only be valid, if it preserves the topology of the domain and of the subcomplex formed by the cricical entities embedded within. As a simple example, an arc connecting two distinct feature nodes may not be assigned a length of 0 as that would either imply degeneration of the map in between or a contraction of the feature's embeddings onto one another—both of which would be unacceptable in the result. We refine the notion of which kind of contractions are forbidden in the next section.

Embedding collapse An integer-grid map with guaranteed valid integers can be computed just from the T-mesh quantization q without actually performing any collapses. This can be done by deriving constraints from q that pin down exactly the integer variables relaxed within the seamless map ϕ and re-solving the global parametrization optimization with these constraints for a quantized map ϕ_q, which is discussed in chapter 8. However, due to the difficulty of domain-generic volumetric parametrization, this global re-parametrization can fail to find an injective map, even though one is guaranteed to exist. While so far the collapse of T-mesh cell embeddings served mainly as a mental image to support intuition for quantization validity, it is feasible to actually perform such collapses, if attention is paid to preserving the validity of the T-mesh in the process. In chapter 9 we therefore carefully design volumetric T-mesh collapse operations, so that the mental image of removing zer-extent cells can be applied in practice. The collapses then yield a positive-only quantization on the remaining T-mesh, for which a block-wise parametrization strategy with injectivity guarantees can be devised.

7.2.4 Structure Preservation Constraints

To define what structure preservation on a T-mesh actually means, we must first define what the critical structure of it even entails. Loosely speaking, we mean the combination of domain topology and the topology of critical entities embedded within the domain. The former we consider characterized by in-contractible cycles and spheres in the domain, e.g., a loop spanning the handle of a torus or a sphere enclosing an inner cavity in the domain. If either were to contract in parameter space, because this is implied by the quantization integers q, this would result in the loss of topological features of the domain. The latter is characterized similarly by incontractible curves and surfaces (that may or may not be closed), e.g., a feature curve, a curve connecting two separate singular links or a boundary region of the domain. If either were to contract in parameter space, this would result in the loss of user-specified (or geometric) features. Below, we make the notion of critical entities and incontractible curves and surfaces explicit.

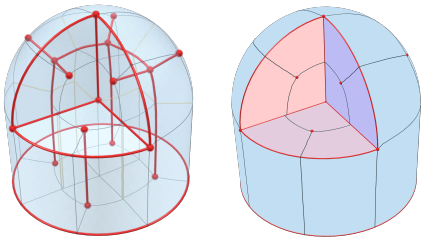
Notation 7.4 (Critical cells). The sets of critical nodes, arcs and patches are denoted $N_!$, $A_!$ and $P_!$, respectively and their union as $C_!$. We refer to the restriction $\mathcal{T}|_{C_!}$ as the *critical submesh* of $\mathcal{T}$, and denote it $\mathcal{T}_!$. In our setting, we assume markings of critical cells to be inherited from the background domain $\mathcal{M}$ as defined in theorem 5.2 (via the embedding $\mathcal{I}$). As the T-mesh construction routine guarantees that critical background entities are captured by discrete elements in the T-mesh (cf. section 5.5.2.4), preserving $\mathcal{T}_!$ implies the preservation of $\mathcal{M}_!$ as well.

Definition 7.5 (Critical T-mesh entities). We define the following high-level critical entities of a T-mesh:

0D Critical nodes are nodes marked as critical.

1D Critical links are maximal chains of arcs marked as critical, unbroken by critical nodes.

2D Critical regions are maximal connected sets of patches unbroken by critical links.



Critical links $l \subset A_!$ are subsets of critical arcs; the set of critical links is denoted $L_! = (l_1, l_2, \dots)$. Critical regions $r \subset P_!$ are subsets of critical patches; the set of critical regions is denoted $R_! = (r_1, r_2, \dots)$.

As all critical entities have a fixed spatial location and extent, we must respect them in the contraction process implied by a quantization involving zero-lengths. We may contract *onto*

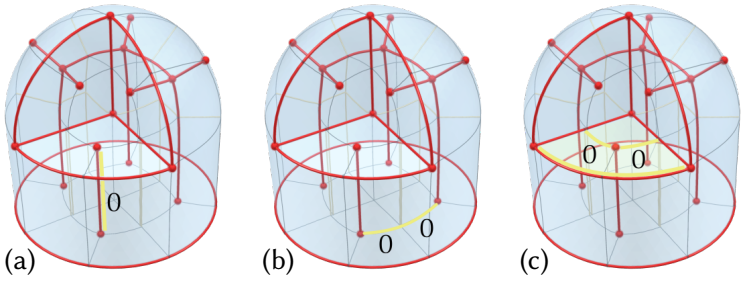


Figure 7.3: Invalid T-mesh quantizations. Singular links and singular nodes are marked in red, regular arcs in light grey. Under the indicated local quantizations (a) the yellow singular link would collapse to a point; (b) two singular arcs would merge along the yellow path; (c) a boundary region would collapse.

them (or along/within them), but must not move them or shrink them (e.g., a curve to a point). A connected component of zero-blocks may therefore be non-collapsible due to involving multiple critical entities, or due to containing one entirely. Figure 7.3 illustrates this. In the following we describe how to preclude such quantizations—and thereby address the main challenge of the problem at hand.

7.2.4.1 Domain Topology Preservation

Let us first derive a sufficiently specific topological invariant of our volumetric domain $\mathcal{T}$ (coupled via $\mathcal{I}$ to the background domain $\mathcal{M}$). If we manage to preserve that invariant under collapses implied by the quantization, we can consider also the overall domain topology to remain unchanged. A quantization length of 0 assigned to an arc or arc path effectively means a contraction of its two end points onto one another in parameter space. If this were to happen to an (arc) loop that is topologically *incontractible* (in $\mathcal{T}$ or in $\partial\mathcal{T}$), a topological mismatch between object and parametrization would be implied, as object and T-mesh (after collapsing 0-elements) were no longer homeomorphic. An example would be the loop going about the volumetric handle of a topological torus. A similar argument can be made about *incompressible spheres*, like one that surrounds a three-dimensional hole in the volume (i.e., an inner void or cavity). The topological invariants implicitly considered here, are the first two homotopy groups of $\mathcal{T}$ and $\partial\mathcal{T}$, $\pi_1(\mathcal{T})$, $\pi_2(\mathcal{T})$, $\pi_1(\partial\mathcal{T})$ and $\pi_2(\partial\mathcal{T})$, respectively. Higher homotopy groups (based on n-spheres with $n > 2$) are trivial for 3-manifolds embeddable in $\mathbb{R}^3$, which is our setting here. Hence, preventing the contraction of non-trivial loops and spheres is sufficient to restrict zero-length assignments during quantization to topology-preserving ones.

7.3 Efficient Constraint Formulation

Pursuant to the above discussion, the following situations must be prevented.

i) A critical link (corresponding to a 1-manifold curve with or without boundary in $\mathcal{T}$) must not be contracted to a point.

ii) Two points on critical entities (distinct or connected via a self-loop) must not be contracted, except along paths lying *completely within* the union of the lowest-dimensional critical entities containing them.

iii) A critical region (corresponding to a 2-manifold surface with or without boundary in $\mathcal{T}$) must not be contracted to a point or curve.

iv) Cycles and spheres that are topologically non-contractible within $\mathcal{T}$ or within its boundary and features must not be contracted.

Some conditions among these, like (i), are compact enough to be set up a priori. The remaining conditions, especially (ii), generally are rather prohibitive to exhaustively enumerate; technically, (ii) implies a constraint per pair of critical points and per (non-redundant) incontractible path between them, which even in a coarse T-mesh could still be a huge number. We therefore suggest an iterative approach: It starts with an under-constrained problem setup, repeatedly solves this problem, and in the process iteratively detects violations of the above conditions, and precludes them in subsequent iterations via additional constraints, until convergence to a truly valid solution. We detail what is meant by a violation below.

7.3.1 Local q-Charts

For any restriction $\mathcal{T}|_{B'}$ of $\mathcal{T}$ to a simply connected, regular subset of blocks $B' \subset B$ (not containing any internal singularity), a quantization q implies an image in $\mathbb{R}^3$ in which each block is a cuboid of the size specified by its arcs quantized lengths q_a. Essentially, q implies the image of a quantized polycube map (section 2.3.7.6) within such a restriction. The image is unique (up to a global rigid transformation): Map one block corner node to $(0, 0, 0)$, and its three incident arcs such that they each align with one of the three principal parametric axes. The remainder is implied by the arc lengths q_a of q. We call such a local chart, with implied integer-grid coordinates per node and integer-grid alignments per arc and patch, a q-chart. The crucial part is that absence of interior singularities eliminates directional and hence positional ambiguity; three spatial directions and coordinates are enough to describe regular space.

In the following we often argue about paths of arcs, and about the relative orientation of a path's arcs and the relative position of nodes. Because we can guarantee that paths always remain confined to simply connected, regular restrictions $\mathcal{T}|_B$, this is to be understood as being meant in the q-chart of this restriction. As only *relative* aspects are considered, the above mentioned rigid transformation is irrelevant, and the notion of arcs of a path being parallel or perpendicular is well-defined. We call a path a zero-path (or 0-path) if its endpoints have distance zero in a q-chart. Under the integer-grid map implied by q the end-points of any zero-path parametrically coincide. This can only encode a valid map when the entire path could be collapsed within the T-mesh to a single point without this violating conditions (i) to (iv). Otherwise, the zero-path signals a violation; we call it a *critical path*. Detection of violation in an intermediate solution q is performed exactly by finding among any the zero-paths of q ones that are *critical ones*. Resolving a violation simply requires adding a constraint, that forces the critical path to expand to greater-than-zero extent. The following explains how to discern critical paths and how to set up the corresponding constraints.

7.3.2 Type (i) Conditions: Link collapse prevention

A critical link $l \in L_!$ would collapse entirely iff all its arcs have length zero under the quantization. To prevent the collapse, we simply need to require

$$\forall l \in L_! : \quad \sum_{a \in l} q_a > 0. \tag{7.5}$$

7.3.3 Type (ii) Conditions: Critical Entity separation

To detect a violation of this condition, we search for a concrete zero-path that is evidence of this. Let p_i be a critical point, i.e., a point contained in one or more critical entities. For instance, a point on a critical link that bounds a critical patch technically lies in both. Let c_i denote the lowest-dimensional of these critical entities; this may be a single critical region, link or node. We

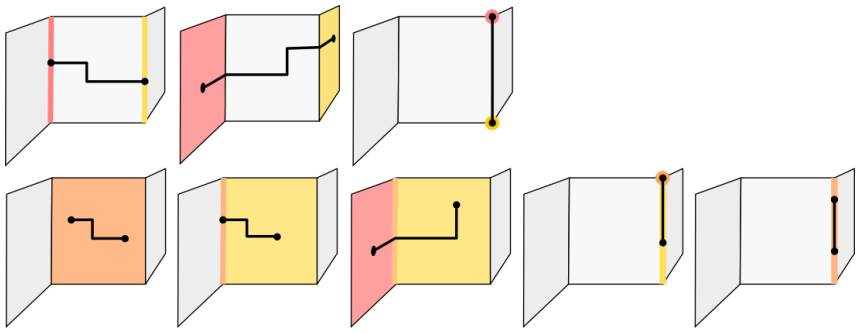


Figure 7.4: Examples of paths between two points on critical entities. Shown are three square boundary regions with in-between vertical critical links. The critical entity c_1 of the start point is marked in red, that of the end point c_2 in yellow, their intersection in orange. Assume the shown paths are zero-paths under q. Then the top row shows violating paths, while those in the bottom row, lying in $c_1 \cup c_2$, are not violating.

search for a path ρ between any two critical points p_1, p_2, that is a zero-path and not contained (under q) in the union $c_1 \cup c_2$, see fig. 7.4.

The rationale for this is as follows: A zero-path between two critical points is evidence that they will become coincident in the zero-block collapsing process mentioned above. This can be validly possible or not. c_i is the zone within which p_i may conceptually be moved in the collapse process without it leaving the critical entities it lies on. If the points need to be moved only within their respective c_i, the collapse is validly possible. Therefore only zero-paths ρ not homeomorphic to those contained in $c_1 \cup c_2$ are critical (homeomorphism here including singularities of $\mathcal{T}$). Hence, a path may well leave the region $c_1 \cup c_2$ (without encircling a singularity or topological feature) and return, but still be non-critical, due to being equivalent to a path that runs within $c_1 \cup c_2$. For arc-paths contained in a (regular) q-chart this notion is easily concretized: Critical paths between c_1 and c_2 need to be (weakly) monotone in coordinates that are not the aligned coordinates of c_1 or c_2. Under this restriction, paths can not leave and return to $c_1 \cup c_2$ via a U-turn. [1]

In section 7.4.1 we describe how to discover exactly such weakly monotonous paths between critical entities. Note that this path discovery is much more efficient than exhaustively enumerating all pairs of critical points and all paths between them: First of all, because only zero-paths are considered; second of all, because only weakly monotonous paths are considered; and third of all, because we confine the search to simply connected, regular q-charts which only exist between singularities (never surrounding them)—but are exactly large enough to contain any critical paths between nearest neighbours. The separation of nearest neighbors then transitively implies pairwise separation of all critical entities.

Constraint formulation For a found critical path ρ between two critical entities c_i, c_j, we add a constraint as follows to prevent the violation in the next iteration. Let $\rho^\perp \subseteq \rho$ be the subset of arcs that are perpendicular to both c_i and c_j (in a common local q-chart along ρ). $\rho^\perp$ may contain arcs aligned with:

- Three axes if c_i, c_j are critical nodes,
- two axes if c_i, c_j are parallel critical links or one is a critical link and the other a critical node,
- a single axis if c_i, c_j are perpendicular critical links or either is a critical region.

[1] While for convex polygonal subdivisions of the plane (such as a 2D T-mesh) a weakly-monotonous arc-path between any two nodes always exists [Dumitrescu et al. 2013; Campen et al. 2015], no such proof is known for convex polyhedral subdivisions of $\mathbb{R}^3$. We conjecture that such a path, however, still always exists within a 3D T-mesh restricted to a q-chart.

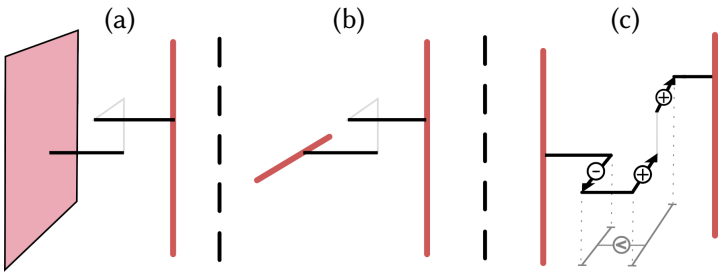


Figure 7.5: Examples of arc paths ρ between critical entities: (a) region and link, (b) two perpendicular links, (c) two parallel links. Arcs that are in $\rho^\perp$ are shown bold. Their sign in constraint (7.6) is positive by default, except for arc sets that are dominated by longer, oppositely oriented arc sets, as in (c).

The following constraint ensures separation of the critical entities in the next iteration, by enforcing a non-zero parametric distance of the two critical points in (at least one) direction perpendicular to the critical entities:

$$\sum_{a \in \rho^\perp} \operatorname{sign}(a, \rho) q_a > 0. \tag{7.6}$$

The restriction to the perpendicular direction(s) is important, because a distance in parallel direction would just shift (non-pointlike) critical entities in parameter space along each other; the coincidence would merely move to another pair of points on the same entities.

The coefficient $\operatorname{sign}(a, \rho) \in \{-1, 1\}$ is chosen positive if a is of *dominant* orientation in ρ, negative otherwise. An arc is of dominant orientation if the total target length (measured via ℓ) of (directed) arcs in ρ pointing in equal direction is greater than the total target length of arcs pointing in opposite direction, see fig. 7.5 right. In case of equality, one orientation is arbitrarily chosen as dominant.

Notice that this constraint flexibly permits separation in either of the two perpendicular dimensions in the case of two parallel singular links, as in fig. 7.5 right—the solution space is not constrained by fixing a separation dimension. Furthermore, the above sign definition permits preserving the original relative parametric location of the two singularities; the opposite choice would enforce "switching places" in the respective separation dimension.

7.3.4 Type (iii) and (iv) Conditions: Topology preservation

In the setting discussed here, where both singularities and boundaries are critical entities that are already prevented from collapsing or merging, it turns out that this also ensures the topology preservation of the latter in all cases—except for a single contrived counterexample. This is captured in the following proposition:

Proposition 7.6 (Sufficiency of type (i) and (ii) conditions). *If all boundaries and singularities are considered critical, enforcing conditions (i) and (ii), indirectly satisfies also (iii) and (iv) unless $\mathcal{T}$ is completely regular. The only connected 3-manifold that is completely regular and embeddable in $\mathbb{R}^3$, is a hollowed solid torus (cf. section 5.5.2).*

A rigorous, low-level proof for the above proposition is hard to formulate. However, we offer the a high-level proof outline in the following. Our experiments on multiple large datasets containing shapes of widely varying topologies support the proposition as well; zero topological defects occurred when employing only constraints of type (i) and (ii) for quantization. The special case of a hollow torus is easily handled by marking an arbitrary node on one of its two boundary components as critical. Self-separation of this node along the two incontractible cycles of the torus ensures a valid quantization; its two boundary components are separated anyway. Note that it is not conceptually hard to incorporate (iii) and (iv) conditions explicitly; in fact, we

demonstrate how to do it in a more general setting (where singularities are not considered critical) in chapter 10, where it is actually required.

Proof outline: (iv) redundant

Assume that $\mathcal{T}$ contains an incontractible loop S^1 that is a 0-loop under the quantization q, but this does not violate conditions (i) and (ii). S^1 does not bound a topological disk within $\mathcal{T}$, otherwise it would be contractible along that disk; instead (in a 2D cross section view) it must bound a disk with one or more holes (representing parts of the boundary $\partial\mathcal{M}$). If it would bound two or more holes, this would imply the contraction of different parts of $\partial\mathcal{M}$, precluded by condition (ii). Also, S^1 does not enclose two (or more) singularities, their implied contraction would similarly be precluded. Hence, S^1 bounds a disk with one hole and up to one singularity.

Assume first that there is no singularity. Due to no singularity being present, S^1 is homeomorphic to (or implied by) a zero-loop that is straight and aligned with a principal axis with respect to ϕ (or an equivalent labeling $\mathcal{L}$ of $\mathcal{T}$). To form a volumetric topological feature of $\mathcal{T}$ it must be part of an entire bundle of such parametrically straight loops that are homeomorphic. Consider now the parametrically orthogonal 2D cross section of such a parametrically straight bundle. It corresponds to the 2D parametrization of a genus 0 surface with boundary. According to the Gauss-Bonnet theorem (or the Poincaré-Hopf theorem for an equivalent cross field [Ray et al. 2008]) it must contain point-singularities (that would be singular loops in the bundle), unless it is an annulus. The bundle can not contain singular zero-loops, as that would be precluded by (i). Hence the cross-section must be an annulus, which when swept along a parametrically straight loop, yields the completely regular hollow torus (that has no singularities).

Assume now that there is a singularity enclosed by S^1 together with the hole. Either it lies in the boundary, or there is a tighter (homeomorphic) 0-loop on the boundary that does not enclose the singularity and for which the above argument holds. In the case not covered yet, where the singularity lies in the boundary, there is a tighter (homeomorphic) 0-loop in the boundary that exactly contains the singularity. Via this loop, the singularity is connected to itself via a parametrically straight path. The (self-)separation of singularities along such paths is included in our notion of condition (ii), hence precluded.

Next, assume that $\partial\mathcal{T}$ contains an incontractible loop S^1 that is a 0-loop under the quantization q, but this does not violate conditions (i) and (ii). If S^1 is incontractible also by passing through the interior $\mathcal{T}$, it is covered by the above argument. Otherwise, S^1 being contractible through $\mathcal{T}$ implies it bounds a disk within $\mathcal{T}$. For a disk, the Gauss-Bonnet theorem demands at least two singularities (totaling an angle defect of $-360°$), the separation of which would preclude the collapse of Ω.

Assume otherwise, that $\mathcal{T}$ contains an incompressible sphere S^2; the Gauss-Bonnet theorem for S^2 demands the presence of point-like singularities (volume singularities piercing the sphere) of total angle defect $-720°$, realizable only with three or more singularities, which means their separation already precludes the collapse of an incompressible S^2.

Alternatively, assume that $\partial\mathcal{T}$ contains an incompressible sphere S^2, compressed under q and not prevented by (i) and (ii). Compression within S^2 is precluded by the above argument, but we can consider compression along its 'diameter'. If S^2 lies within an outer, sphere-topology boundary component, Gauss-Bonnet implies that the interior of the volumetric domain must contain volume singularities and/or inner boundaries. Hence, the compression along its diameter is precluded because the boundary as a critical region is separated via (ii) from anything enclosed within it. If S^2 instead lies within an *inner* sphere-topology boundary component enclosing a

ball-topology void, there are no quantization variables within that hole that could even imply the compression of the sphere along its diameter.

In summary, except for the hollow torus, the collapse of incontractible loops S^1 and spheres S^2 within either $\mathcal{M}$ or $\partial\mathcal{M}$ is prevented by conditions (i) and (ii).

Proof outline: (iii) redundant

Assume now that $\mathcal{T}$ contains a critical region r (e.g., a boundary component), which is an iso-parametric surface, whose collapse is implied by the quantization q, but this does not violate conditions (i), (ii) and (iv). The region r can not be bounded by multiple critical links aligned to different principal parametric axes, as that would imply collapse of these links, precluded by (ii).

Assume, that r is bounded by one or more links, chains of which are parametrically straight, then (i) guarantees the circumference of r does not collapse. Hypothetically, r could still collapse along its 'interior', but only if it does not contain two or more singularities or is bounded by two or more components, otherwise (ii) would already have precluded it. Due to this, r being a disk or annulus is ruled out (the only genus 0 cases). In case of genus ≥ 2, the interior of region r would contain at least one point-like singularity and parametrically straight loops that are incontractible within r; (self-)separation of the contained singularities along such cycles implies that the interior of r can not collapse. In case of genus 1 there is no interior singularity; however, there would be a parametrically straight path connecting the boundary of r to itself through r's interior; then, self-separation implies that the interior can not collapse.

Assume now, that r is closed, i.e., not bounded by links. Then, the Gauss-Bonnet theorem demands 3 or more singularities for the sphere (totalling an angle defect of $-720°$), the pairwise separation of which would prevent collapse of r within the surface. Compression through the volume enclosed by a sphere is precluded via the above argument for incompressible spheres.

Assume r is a closed surface of genus ≥ 2, hence must contain one or more singularities. Any cycles incontractible within r are prevented from collapsing through (self-)-separation of singularities with an argument identical to that for condition (iv). Hence, also r can not collapse.

An iso-parametric genus 1 surface, i.e., a torus, could have no singularities. Its fundamental group is formed by two classes of incontractible generator loops S^1; such that wrap once around its handle and such that wrap once around its hole. If either is contractible when allowing to pass through the entirety of $\mathcal{T}$, it bounds a disk, and because a disk implies two or more singularities enclosed by S^1, (ii) prevents this from being a zero-loop. Otherwise, S^1 is also an incontractible loop of $\mathcal{T}$ and the argument for condition (iv) applies; either r (via its incontractible generator loops) is already prevented from collapsing, or it is one of the two torus-topology boundaries of the hollow solid torus.

7.4 Lazy Constraint Strategy

Initially, we use only constraints (i); these are inexpensive to set up, so it is sensible to add all of them to the IQP from the start. Constraints of type (ii) are added lazily only when examination of the previous IQP solution reveals concrete violations. A subset of the violated constraints is then made explicit and added to the IQP, which is then resolved (with warm start). This is iterated if necessary.

This lazy approach allows for an efficient processing. In particular, in cases that are conflict-free right away, a single solve (with a low number of inequalities) is sufficient. Only in harder cases (roughly those that the classical rounding approach would likely fail on) additional effort (setting up further constraints, resolving the extended IQP) needs to be spent. Due to the careful

way we select the lazy constraints, the number of iterations needed is small even in extreme cases, see section 7.6.2.

7.4.1 Discovering Critical Paths

The set of critical points is infinite. We therefore consider critical nodes, and also entire critical arcs and critical patches at once, and determine whether a zero-path exists between any two points on any pair of these.

Searching the Relevant Space To this end, for each critical node, arc or patch c and for each outgoing parametric direction d perpendicular to it, we construct a spanning forest $Y(c,d)$ in the graph of arcs A, rooted at the nodes of c_i. Note that a critical region has just two perpendicular directions (one if in the boundary), while there are k outgoing directions at a critical link of valence k, and potentially more at critical nodes.

Let $B^0(c)$ denote the set of blocks of $\mathcal{T}$ that, under q, have zero distance to c, i.e., those that after collapsing all zero-blocks would intersect c. We can restrict the construction of $Y(c,d)$ to $B^0(c)$, as any relevant critical path ρ necessarily is contained in this zone. We furthermore stop the construction as soon as the first critical entity c' is reached in such a way that there is a path ρ in $Y(c,d)$ that satisfies the criteria of violation spelled out in section 7.3.3. The rationale is that enforcing parametric separation just between nearest neighbors, due to the transitive effect of the inequality constraints, often is already sufficient to yield a valid quantization. If not, further violations will be detected and constraints be added in the next iteration.

We construct the forest $Y(c,d)$ as a forest of shortest paths in ϕ. Arcs are conquered, starting from all nodes of c, in a Dijkstra-like manner, restricted to arcs on blocks from $B^0(c)$, and never walking in direction $-d$. The resulting spanning forest therefore is formed by (weakly) d-monotone paths. Figure 7.6 illustrates such forests.

Region regularity During search, singularities can be encountered. Once a conflicting critical entity c' (possibly a singularity) is reached in direction d, the forest $Y(c,d)$ is not extended any further. Hence the interior of the expanded part of B^0 does not contain a conflicting singularity. A non-conflicting singularity c' could be part of B^0 in one of two ways: It can be non-overlapping with c under the q-chart, in which case it must lie at the boundary of B^0, or it can have an aligned

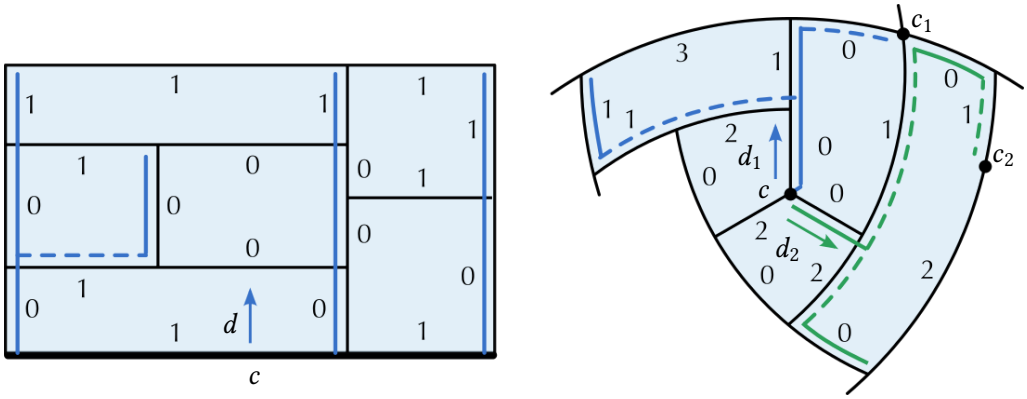


Figure 7.6: Forests $Y(c,d)$ in $B^0(c)$ in a perpendicular cross section view of (left) a boundary region (bold curve), and (right) a critical arc (central dot). On the right, two forests (in this case trees) for two different directions d_1, d_2 are shown (blue and green). Forest arcs that are not parallel but perpendicular to its defining direction are dashed. On the right, note that the encountered singularity c_1 is not in conflict with c (i.e., $\phi_q(c) \cap \phi_q(c_1) = \varnothing$), but c_2 is.

direction that is parallel to d. In the latter case c' lies in the interior of the expanded part of B^0, and must directly emerge from c along d. Otherwise a critical node that bounds c', would have been expanded first and the search stopped due to the conflict. When a non-conflicting singularity c' emerges from c along d, the choice of a q-chart along it creates ambiguity, but only for the non-aligned directions of c', never for d. In such cases, ambiguity can still be avoided: If a potentially conflicting critical entity is encountered in $Y(c, d)$, overlap is checked in each of at most 4 unique q-charts around c', and—if a conflict is encountered in any such chart—only the d-coordinate is used for constraint setup. This is sufficient, because d (as the weakly monotone coordinate) is guaranteed to provide at least one arc along which separation can be enforced. In all cases other than the one just described, the forest remains contained in a regular subset of B^0. Then, our notion of a single non-ambiguous q-charts holds, in which all directions and relative coordinates of entities within B^0 are uniquely defined.

Checking for a Conflict When a critical node n', an arc a' of a critical link or a patch p' of a critical region is encountered (checked in this priority order), these are treated as representatives of their containing critical entity c'. If d is perpendicular also to c', we check whether the images of c and c' intersect in the common q-chart of B^0. This would imply that a point of c and a point of c' are contracted together along the arc-path ρ under the metric implied by q.

To this end, recall that critical arcs and patches are *aligned*. Hence, their image under ϕ as well as under any quantized parametrization ϕ_q consistent with q is an axis-aligned one- or two-dimensional box (a line segment or rectangle). Let $\phi_q(c')$ denote this box of c' in a local q-chart along ρ. We check it for intersection with $\phi_q(c)$ in the common chart. If they intersect, the critical entities are in conflict, the criteria of violation are satisfied, so a lazy constraint of type (7.6) is added for the next iteration. Note that paths contained in $c_1 \cup c_2$ (under q, which would not be violating, cf. section 7.3.3) are not discovered because the forest does not have arcs within c, and the above restriction to weakly d-monotone paths prevents a return to c within B^0.

Figure 7.7 sketches the iterative enforcement of separation between initially conflicting entities. Figure 7.8 shows a volumetric example.

7.4.2 Constraint Feasibility

Analogous to the argument in appendix A.2 of [Campen et al. 2015], an assignment of strictly positive integers that are consistent exists: Scaling the rational parametric lengths of arcs in the input parametrization by their lowest common denominator s yields one example. For this

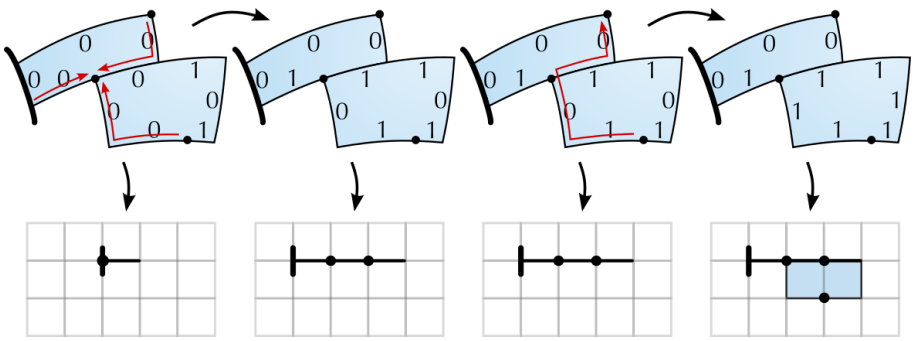


Figure 7.7: Shown is an excerpt from a larger T-mesh, together with a quantization q, and an image under q (bottom). One critical link is involved (bold black curve), and three critical nodes or perpendicular links (black dots). From an initial, invalid state (left), a valid quantization (right) is derived by, here in two iterations, finding critical paths (red) and "inflating" along them using constraints (7.6) in the subsequent solution.

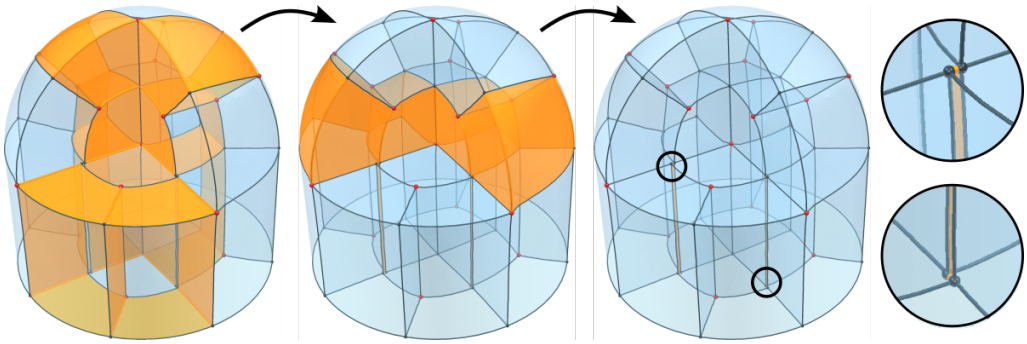


Figure 7.8: An invalid quantization evolves due to the addition of constraints in two iterations. Marked in orange are arcs of length zero and patches of area zero under the quantization. In the end, four zero-arcs and two small zero-patches (see blow-ups) remain which cause no further conflicts.

assignment q^*, as all arcs have a positive value, all types of lazy constraints are satisfied: For those with possibly negative coefficients (7.6) this may not immediately be obvious; however, note that the positive coefficients are associated with the arcs of dominant orientation (with larger total length in ϕ, thus also in $s\phi$ and under q^*). Therefore, the IQP remains feasible no matter what subset of lazy constraints is added in what order.

7.5 Algorithm Summary

The pseudocode in algorithm 7.1 gives a concise summary of the described quantization method as a whole. The scaling factor s in the objective function is a parameter that allows choosing the targeted grid or mesh resolution. Its application to target lengths ℓ is equivalent to globally scaling the input seamless parametrization ϕ by s, increasing its volume and thus the target number hexahedra by factor s^3.

7.5.1 Implementation Notes

Our reference implementation is publicly available as the open source library QGP3D[2]. It is based on the MC3D library and employs the motorcycle complex construction routine from the previous chapter, configured so that only regular-reductions are applied. Then, any critical entity has patches and arcs in each outgoing parametric direction, allowing for an efficient traversal of forests $Y(c, d)$ by demanding that the first traversed arc must be aligned with d and skipping any nodes outside of c that have been visited by shorter paths. It is important to skip only nodes outside of c, because self-separation of c along topologically non-trivial cycles must still be assured.

As solvers, tasked with the repeated solution of the central integer quadratic program (7.1), our software allows choosing between a commercial generic branch-and-cut solver ([Gurobi Optimization, LLC 2025]), an open-source branch-and-cut solver ([Bonami et al. 2008]) or our own open-source solver, specifically tailored to the use case as discussed in chapter 11. For branch-and-cut solvers, run times to find optimal solutions can be unpredictably long, so working with time limits is recommended. A strategy we employ, is to solve with a rather short time limit by default and employ a more thorough solve once an intermediate solution generates no more additional constraints.

The core quantization algorithm can be implemented on any abstract T-mesh (theorem 6.3) completely independent of any embedding into a background domain. Singularities can be deduced

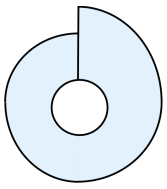
[2] https://github.com/HendrikBrueckler/QGP3D

Algorithm 7.1: Volumetric Quantization

Input: mesh $\mathcal{M}$, valid seamless parametrization ϕ
Output: valid quantization $q_a \in \mathbb{Z}$ on $\mathcal{T}$ of $\mathcal{M}$

$\mathcal{T} \leftarrow$ volume-T-mesh$(\mathcal{M}, \phi)$ // motorcycle complex
$\ell_a \leftarrow \|\phi(a)\| \in \mathbb{R}, \forall a \in A$ // consistent input arc lengths
foreach patch $p \in P$ **do**
  for $i \in \{1, 2\}$ **do**
    add constraint $\sum_{a \in A_i(p)} q_a = \sum_{a \in A_{i+2}(p)} q_a$ // (7.3) rectangular patches
* **foreach** block $b \in B$ **do**
  for $i \in \{1, 2, 3\}$ **do**
    add constraint $\sum_{a \in A_i(b)} q_a \geq 0$ // (7.4) non-negative blocks
foreach critical link $l \in L_!$ **do**
  add constraint $\sum_{a \in l} q_a > 0$ // (i) link non-collapse
repeat
  solve $\sum_{a \in A} (q_a - s\ell_a)^2 \rightarrow \min,\ q_a \in \mathbb{Z},\ s.t.$ constraints
  foreach critical entity c **do** // (ii) critical separation
    foreach direction $d \perp \phi_q(c)$ **do**
      while expand d-monotonic forest $Y(c, d)$ within $B^0(c)$ of ℓ-shortest arc paths **do**
        $\rho \leftarrow$ path from c to p' in $Y(c, d)$
        $\phi_q \leftarrow$ common q-chart(s) along ρ
        if p' on critical entity c' with $\phi_q(c') \perp d$ **then**
          $\rho_\perp \leftarrow \{a \in \rho \mid \phi_q(c) \perp \phi_q(a) \perp \phi_q(c')\}$
          if $\rho_\perp \neq \emptyset$ and $\phi_q(c) \cap \phi_q(c') \neq \emptyset$ **then**
            add constraint $\sum_{a \in \rho^\perp} \text{sign}(a, \rho) q_a > 0$
            break
until no new constraints were added
return integers q_a on $\mathcal{T}$

from angles in the abstract T-mesh, feature markers can be freely prescribed and target lengths ℓ can be chosen sensibly. It is, however, necessary to demand that the T-mesh either has a structure that admits an assignment of positive integers that is consistent (2D counterexample in the inset), or alternatively allows for certain topological adjustments implied by q (assigning a 0 to the step on the outer boundary in the inset). As argued in section 7.4.2 a consistent positive integer assignment is always possible in our case, i.e., for T-meshes built on seamless parametrizations. Hypothetical future scenarios, involving inconsistent 3D T-meshes from other sources, could benefit from T-mesh repair operations known from surface meshing [Usai et al. 2015; Campen and Zorin 2017].



7.6 Results: T-Mesh Quantizations

Here, we first consider only aspects of the produced T-mesh quantizations themselves. For this evaluation we used the commercial branch-and-cut solver to solve (7.1) [Gurobi Optimization, LLC 2025], configured with a time limit of $1h$ which was reached in a handful of cases (cf. section 7.6.2.1), at which point the solver had already found feasible solutions with optimality gaps of $< 20\%$. The computation of integer-grid maps adhering to the T-mesh's integer arc lengths, as well as results demonstrating such integer-grid maps and resulting hex meshes, are treated in the next chapter.

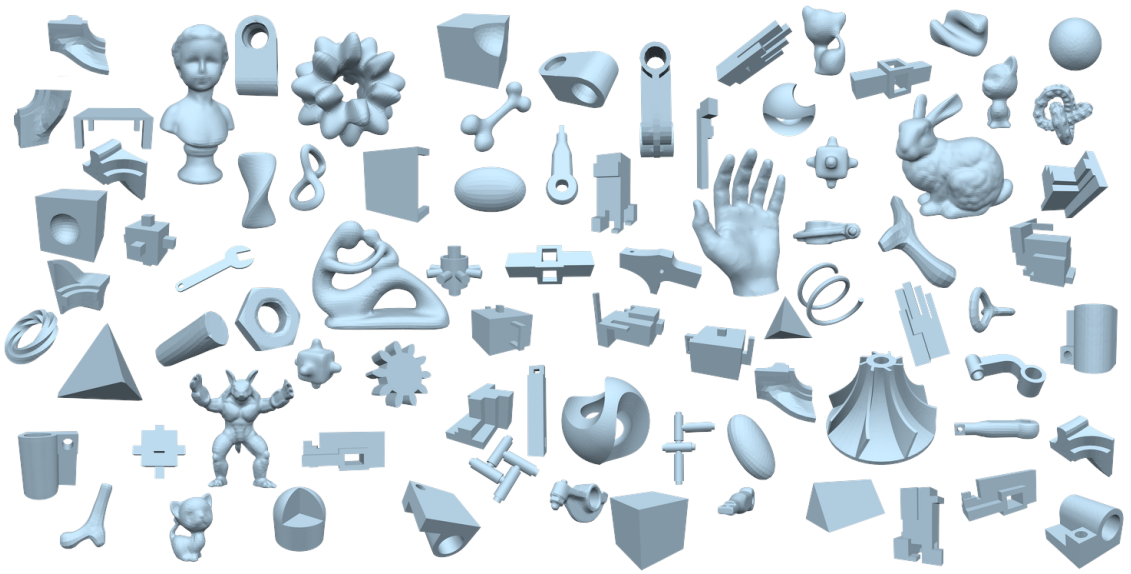


Figure 7.9: Gallery of models from the datasets used for our experiments. Note that some may look like duplicates, but singularity structures differ.

7.6.1 Datasets

As input (fig. 7.9) for our experiments we use 15 seamless (non-integer) parametrizations generated using the approach of [Liu et al. 2018], provided by the authors, and 75 seamless (non-integer) parametrizations we generated using the approach from [Nieser et al. 2011], i.e., with a singularity structure derived from given meta-meshes. Meshes released with a number of papers [Gregson et al. 2011; Li et al. 2012; Livesu et al. 2013; Huang et al. 2014b; Gao et al. 2015b; Livesu et al. 2015; Cherchi et al. 2016; Fang et al. 2016; Fu et al. 2016; Livesu et al. 2017; Shang et al. 2017; Wu et al. 2017, 2018; Cherchi et al. 2019; Corman and Crane 2019; Takayama 2019] served as meta-meshes in this context.

7.6.2 Quantization Behaviour

When aiming to create an extremely dense, high resolution integer-grid map or hexahedral mesh, the choice of approach to decide on integer values is uncritical. Whether T-mesh-based or rounding-based, and regardless of parameter settings, for a sufficiently fine resolution they will eventually yield a valid assignment. Differences therefore become most apparent when targeting coarser resolutions.

We start by considering the extreme case, aiming for as little total parametric volume ("number of hexes") as possible. The rounding-based strategy consistently fails in this scenario, it cannot produce a valid integer-assignment for any input. Our T-mesh-based quantization, even for a scaling factor $s=0$ in algorithm 7.1 yields a valid solution by construction. We compare the effect of supporting either $\mathbb{Z}^{>0}$, $\mathbb{Z}^{\geq 0}$, or $\mathbb{Z}$ as value range of the quantization q. The former (*positive*) is what was used in the proof-of-concept quantization in section 6.4.1. Methods for the 2D case often support *non-negative* values. Our approach even supports *arbitrary* values, subject to non-negative block extents (7.4). Table 7.1 lists the parametric volume of the quantization results for these three options, when aiming for maximum coarseness.

Effect of Zeros Significant differences can be observed in table 7.1. Requiring strictly positive quantization values on arcs leads to results up to (in one case) 23× more complex than the non-negative case. While the coarsest possible result may not be the one needed in concrete use

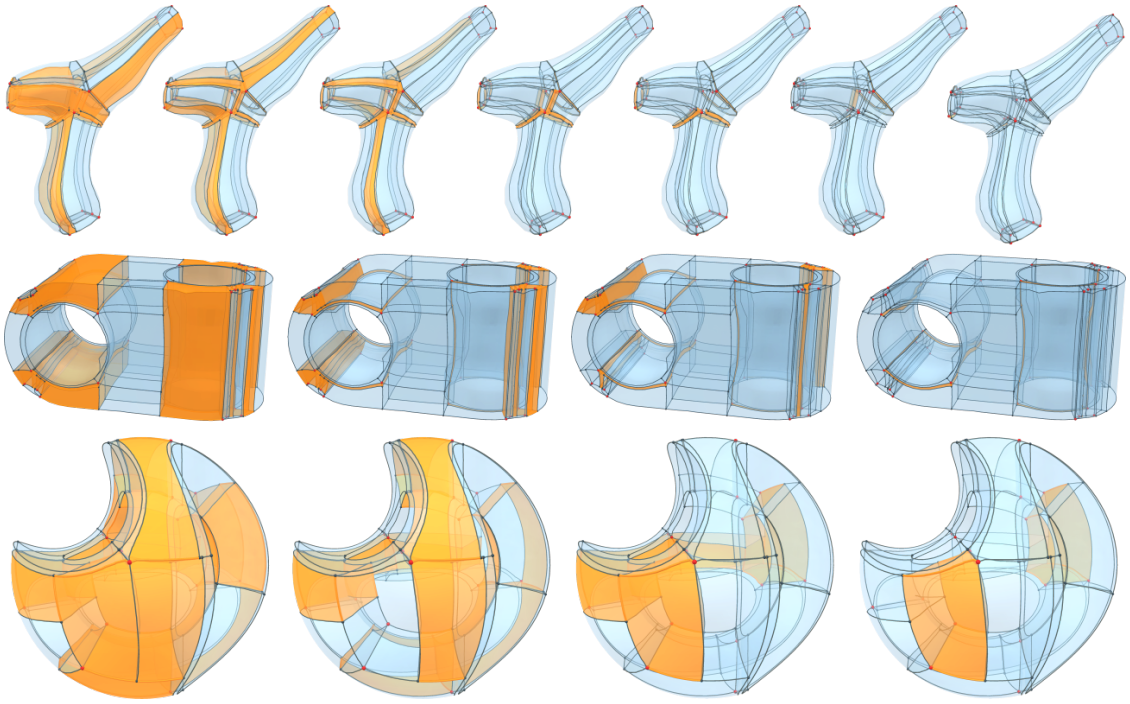


Figure 7.10: Zero-arcs and zero-patches of valid quantizations computed by our method are highlighted in orange in a semi-transparent rendering of the MC. From left to right, the set target resolution is increased, thus the number of elements that are ideally quantized to zero decreases.

Table 7.1: For inputs from our two datasets, the smallest quantized parameter volume ("number of hexes") that is achieved depending on the employed value range ($q_a \in \mathbb{Z}^{>0}$, $\mathbb{Z}^{\geq 0}$, or $\mathbb{Z}$) is shown in grey. The last columns show the ratios of these numbers. (The complete right table was published in the appendix of [Brückler et al. 2022a]).

Model	$\mathbb{Z}^{>0}$	$\mathbb{Z}^{\geq 0}$	$\mathbb{Z}$	$\frac{\mathbb{Z}^{\geq 0}}{\mathbb{Z}^{>0}}$	$\frac{\mathbb{Z}}{\mathbb{Z}^{>0}}$
SCULPTURE	108	16	16	15%	15%
CYLINDER	26	5	5	19%	19%
JOINT	205	62	62	30%	30%
ARMADILLO	1832	596	670	33%	37%
ROCKERARM	1347	483	483	36%	36%
FANDISK	110	51	51	46%	46%
BROKEN BULLET	44	24	24	55%	55%
BONE	87	56	56	64%	64%
CAMILLE HAND	122	79	79	65%	65%
KITTEN	176	139	139	79%	79%
CUBE SPHERE	10	10	10	100%	100%
FANPART	5	5	5	100%	100%
PRISMA	3	3	3	100%	100%
SPHERE	7	7	7	100%	100%
TETRAHEDRON	4	4	4	100%	100%

Model	$\mathbb{Z}^{>0}$	$\mathbb{Z}^{\geq 0}$	$\mathbb{Z}$	$\frac{\mathbb{Z}^{\geq 0}}{\mathbb{Z}^{>0}}$	$\frac{\mathbb{Z}}{\mathbb{Z}^{>0}}$
EXAMPLE 2	16824	732	406	4%	2%
DOUBLE HINGE WH	1826	81	78	4%	4%
CUBESPIKES POLYCUBE IN	736	33	33	4%	4%
CUBESPIKES POLYCUBE OUT	582	33	33	6%	6%
CUBESPIKES MODEL IN	3687	241	241	7%	7%
CUBESPIKES MODEL OUT	3460	334	284	10%	8%
FEMUR SHELL1	190	19	20	10%	11%
FANCY RING-HEX	120	15	15	13%	13%
ROTELLIPSE PADDED	134	19	19	14%	14%
TWISTEDU	134	19	19	14%	14%
SCULPTURE-A	108	18	18	17%	17%
COLUMN	100	18	18	18%	18%
BUMPY TORUS	3927	811	806	21%	21%
EXAMPLE 3	5164	1189	1086	23%	21%
ROD	301	64	64	21%	21%
⋮					

cases, this difference illustrates the significantly higher general flexibility, the significantly larger space of valid integer assignments that can be represented and achieved when supporting zeros. Note also that when targeting the coarsest possible result, the number of implied hexahedra approximately equals the number of blocks in the coarsest block decomposition. Hence, a smaller number of minimum hexes implies that a method can achieve hex meshes with a cleaner block-structure and a simpler (feature-respecting) base complex. These aspects are a major part of the justification for the extra effort which is necessary exactly to enable the safe use of zeros (essentially all of algorithm 7.1 starting from the line marked by $*$, except for the solve). fig. 7.10 provides further insight on this matter, demonstrating that the assignment of zero-lengths is beneficial even for fine target resolutions.

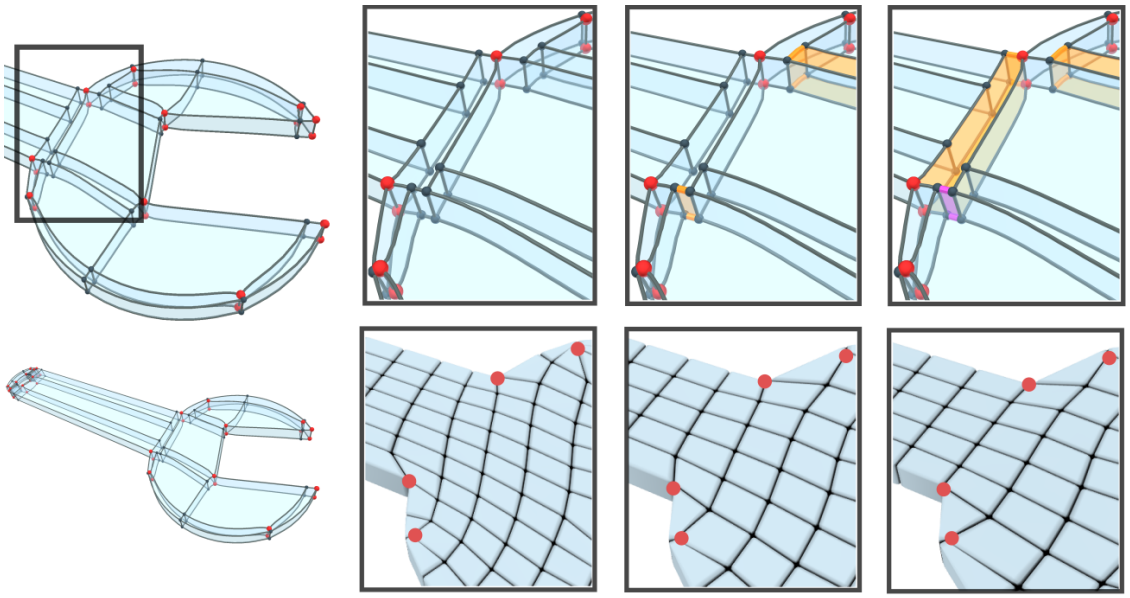


Figure 7.11: Features, such as singular nodes (red), that are not perfectly aligned in the input parametrization may be forced to misalign even more when operating with only positive arc values (left). Supporting zeros (orange, center) or even negative values (pink, right) yields results (bottom) with improved structure.

Effect of Negatives When allowing even negative values, which, to the best of our knowledge, is considered in this context for the first time, even up to 41× simpler results can be achieved relative to the positive case. The benefit relative to the non-negative case in terms of maximum coarseness is less significant, though (a maximum of 1.8× in our tests). But considering that it does not require additional algorithmic effort, it is well worth using, especially because it has benefits in terms of enabling improved structural quality (fig. 7.11).

7.6.2.1 Timing

The main contributors to the run time of our quantization method are the general setup (T-mesh structure assembly, etc.) , the lazy discovery of critical paths, and (repeated) solving of the IQP. The latter, being a discrete optimization problem with numerous equality as well as inequality constraints, may be expected to be the dominating part. However, note that the complexity of the T-mesh $\mathcal{T}$, and therefore the number of variables and constraints, when constructed as a motorcycle complex depends mainly on the complexity of the singularity network. It does not depend on the resolution of the input mesh $\mathcal{T}$. In fact, on average the IQP solution or solutions

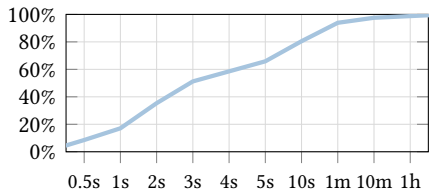


Figure 7.12: Fraction of all input dataset cases for which quantization takes less time than indicated on the horizontal axis. The time taken is linked to the number of arcs in the T-mesh; as a rule of thumb, 15ms per arc can typically be expected.

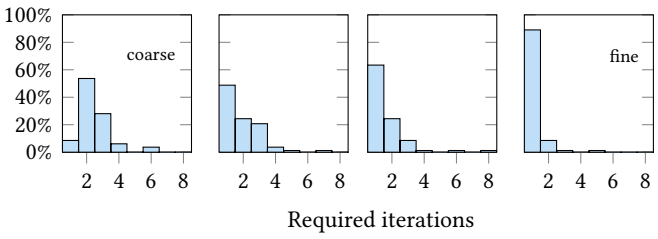
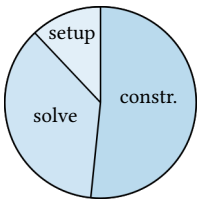


Figure 7.13: Number of lazy constraint iterations needed. Reported are the number n of input cases from the dataset that needed the respective number of iterations. Each histogram shows the numbers for a different level of target coarseness, from maximally coarse (left) to fine (right).



took 36% of the total time (see inset). The main share is commonly taken by the constraint construction by means of critical path discovery. Only occasionally we observe the solver in total clearly dominating the overall run time (in 8% of the cases $\geq 75\%$). Absolute values are indicated in fig. 7.12. Due to our choice of lazy constraint ordering strategy, the number of repeated IQP solves, i.e., of iterations of lazily adding constraints, is typically quite low. As can be seen in fig. 7.13, in the majority of cases one or two solves are sufficient, an initial solve with link constraints of type (i), followed by a second solve with critical separation constraints of type (ii). As expected, coarser target resolution cause more conflicts and require more iterations on average.

7.7 Discussion

The T-mesh quantizations introduced and demonstrated in this chapter induce an integer-grid map with all integer degrees of freedom fixed. This is easiest to see for non-negative integer assignments: One could eliminate all 0-extent cells of the T-mesh by collapsing them in object space. This makes sense intuitively: If they should be assigned a parametric extent of 0, they must also have zero extent in object space, otherwise the map implied by the quantization is degenerated. Our T-mesh quantization routine ensures the existence of a sequence of such collapses, that does not affect critical entities nor the domain topology. In the resulting collapsed T-mesh an integer-grid map can be initialized per block, under which each block is mapped to a parametrically aligned cuboid with integer dimensions prescribed by the remaining positive arc lengths. The block-wise maps together obviously form a global integer-grid map with integer variables fixed by prescribing integer arc lengths, or equivalently the integer positions of nodes.

However, because arcs do not necessarily run only between endpoints on critical entities, they actually also constrain some of the continuous degrees of freedom of the mapping problem. In other words, a map optimization that is confined to preserve the integer coordinates for each node of the T-mesh—even non-critical ones—is over-constrained; we do not actually care whether a 'random' vertex somewhere in the domain is mapped to any specific coordinate. Conceptually, we would like to prescribe only *exactly* the integer degrees of freedom in the global integer-grid map problem, so that all continuous variables remain free to improve the primary goal in eq. (3.5) (typically map distortion).

In the following chapter, we therefore explain how to extract from a given T-mesh quantization—built on a seamless map ϕ—a set of constraints to feed into a global re-computation of a topologically identical but quantized map ϕ_q. These constraints fix exactly the problem's integer variables, and hence are a direct replacement for the fragile rounding previously used in most publications in the field; the remaining problem can be efficiently optimized over a purely continuous domain. Intuitively, the constraints correspond to a spanning tree of prescribed integer spacings and sizes of critical entities, as well as prescribed integer-displacements along topologically non-trivial cycles. Such spacings, sizes and displacements once more can be expressed via arc-paths in the T-mesh, and amount to sums over individual integer-length arc vectors. In the sum formulation, individual negative arc lengths $q_a < 0$ are cancelled out and do not pose a problem.

The global constrained re-computation of ϕ_q, however, introduces another potential robustness gap: The optimization routine might fail to find an injective map even though the chosen integers admit one. This is exactly the reason, why the collapsing routine in chapter 9 is crucial for a fully robust pipeline: It allows a guaranteed-injective initialization of the subsequent optimization.

Global Map Reparametrization

8

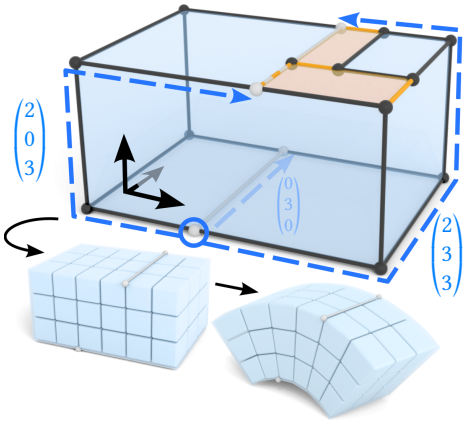
IV Quantize the seamless map ϕ into an integer-grid map ϕ_q.

IVa–c A quantized, ϕ-aligned T-mesh $\mathcal{T}_q$ in the volume

IVd Get constraints from $\mathcal{T}_q$ that force $\phi \mapsto \phi_q$

… Guarantee injectivity of ϕ_q…

V Extract a hex mesh from ϕ_q.



Recall that a seamless map is just the continuous relaxation of an integer-grid map. Going from one to the other then requires primarily to decide on integer values for the relaxed variables, and secondarily, to update the remaining continuous variables to satisfy injectivity constraints and improve map quality. We refer to the whole process as *map quantization*. Here, by transcribing a valid solution of a proxy problem (the T-mesh quantization) to integer values for the map, we demonstrate a robust solution to the primary part. This primary solution is robust in that it guarantees that a valid solution can be found in the secondary part. The rounding procedure employed in established integer-grid map generation pipelines [Nieser et al. 2011; Li et al. 2012; Jiang et al. 2014; Fang et al. 2016; Liu et al. 2018], does not offer this robustness. In fact, if the target hex mesh edge length approaches (or falls below) the local feature size, it inevitably precludes valid solutions for the secondary part. Our quantization method is to be used as a drop-in replacement for this fragile rounding procedure. The main focus therefore is to solve the interfacing question of how to plug our T-mesh based solution into a global parametrization method that:

1) Computes a seamless map ϕ on $\mathcal{M}$ via continuous optimization,

2) calls our method on $(\mathcal{M}, \phi)$, which
   a) internally builds a T-mesh $\mathcal{T}$,
   b) computes a quantization q on $\mathcal{T}$,
   c) …
   d) returns *only* a set of minimal constraints C_q,

3) then restarts the exact same continuous optimization from 1), but additionally subject to C_q, such that the resulting ϕ_q is an integer-grid map.

Below we discuss how to fill in the blank in this process.

8.1 Notation

A seamless parametrization on a tet mesh $\mathcal{M} = T \cup F \cup E \cup V$ is represented as per-vertex parameters $\boldsymbol{u}_v = (u_1^v, u_2^v, u_3^v)$ (more than one per vertex only along the cut surface). Let U denote the set of all vertex parameters. Let the subset $U_! \subseteq U$ denote the set of *critical parameters*; it contains a parameter $u_j^{v_i}$ if its vertex v_i would be required to lie on an integer in coordinate j in an integer-grid map. More specifically, it contains the aligned-coordinate parameters of (non-pointlike) features, boundaries and singularities, as well as all parameters of feature vertices.

Remark 8.1 (Quantized Seamless Map). A map ϕ is an integer-grid map iff it is seamless and additionally quantized, meaning:

- for each translatory component d_f of transition τ_f: $d_f \in \mathbb{Z}^3$, (8.1)
- for each critical parameter $u_j^{v_i} \in U_!$: $u_j^{v_i} \in \mathbb{Z}$. (8.2)

Equivalence We call two integer-grid maps *equivalent* if they imply the same mesh. Note that modifying a map by applying (per tet) an arbitrary integer translation yields an equivalent map, as the grid pre-image is invariant to such translations. This means that $|T| - 1$ of the $|F \setminus \partial\mathcal{M}|$ (three-dimensional) transition translation degrees of freedom d, are irrelevant under equivalence, where $|T|$ is the number of tets and $|F \setminus \partial\mathcal{M}|$ the number of non-boundary facets. For instance, for each facet f crossed by a spanning tree of the tet adjacency graph (a total of $|T| - 1$), we may fix $d_f = 0$ without restricting the space of representable meshes [Li et al. 2012].

Integer Feasibility The remaining $|F \setminus \partial\mathcal{M}| - |T| + 1$ translations, as well as the singularity and boundary related integer degrees of freedom are not independent, but related by the involved conditions (seamlessness and alignment). Overall, the number of independent integers depends on the complexity of the singularity network and the topological complexity of $\mathcal{M}$, not on the number of elements in the mesh. We exploit this in section 8.2.1.

While independent with respect to the linear seamlessness, boundary alignment, and singularity alignment conditions, values of these integers cannot actually occur in arbitrary combinations in integer-grid maps. The reason is the additional (nonlinear) map injectivity condition **IGM$_\mathbf{I}$**. In other words, for each consistent choice of integers, maps exist that are grid-preserving, but not necessarily a valid, injective integer-grid map exists. We call a choice of integer values *infeasible* if it does not permit a valid map (even under mesh refinement).

This is the key limitation of existing rounding-based methods to decide on the integer values [Nieser et al. 2011; Li et al. 2012; Jiang et al. 2014] in the context of integer-grid map generation; they take into account the simple linear dependencies, but ignore the intricate injectivity condition **IGM$_\mathbf{I}$**. They may therefore yield infeasible integers, and especially for coarse target grid resolutions they often do—for a sufficiently coarse target resolution they even necessarily will. This explains why example meshes shown in pertinent articles are sometimes of rather dense, high resolution. The simplest example of a conflict is the choice of coincident integer parameters for two spatially distinct singular or boundary vertices. This implies a parametric distance of zero in any compatible map, such that condition **IGM$_\mathbf{I}$** cannot hold along some path between the two vertices.

The integers produced by our method on the T-mesh, in contrast, are guaranteed to be feasible by construction; the question that remains is how to transcribe them into constraints that fix exactly eqs. (8.1) and (8.2) but no other variables.

8.2 Constraint Extraction from a Quantized T-Mesh

A quantization q on a T-mesh $\mathcal{T} = B \cup P \cup A \cup N$ is an assignment of integer (parametric) lengths q_a to each of its arcs $a \in A$. This needs to be translated into integer parametric coordinates for critical entities and corresponding integer transition translations in a to-be-computed integer-grid map. Intuitively, this can be achieved by "integrating" the quantization values along arcs, starting from some node fixed to the origin $(0, 0, 0)$ and extending this along the quantized arc network, so as to yield fixed integer coordinates for all other nodes. Fixing the translation components of all transition functions additionally requires measuring the integer parametric displacements along non-trivial cycles of $\mathcal{T}$. Setting up a constraint for each individual node and non-trivial cycle of $\mathcal{T}$, and recomputing a seamless parametrization subject to these constraints, would then obviously yield an integer-grid map.

However, this set of constraints would be unnecessarily restrictive (constraining more than just the integer degrees of freedom, also the positions of points that coincide with regular nodes) as well as redundant, though. We can reduce it to a smaller set by constraining only larger *arc paths*, i.e., fixing the integer displacement along each such path, which then only constrains exactly the integer degrees of freedom, as detailed in the following. Afterwards, in section 8.2.2, we will consider factors that complicate the formulation of constraints from given arc paths and lastly, in section 8.2.3, detail a concrete procedure that obtains the final constraints from the set of arc paths.

8.2.1 Choice of Constraint Paths

Consider first a set of path constraints, in which each individual arc is considered a path and represented by an explicit constraint fixing the parametric distance between its endpoints. The network formed by such exhaustive path constraints contains many contractible cycles. Each such cycle represents constraint redundancy, as decomposing the cycle into two noncyclic paths yields one constraint path that is already implied by the other path and rectangularity constraints eq. (7.3). Therefore, instead of all arcs, it suffices to constrain an arc-based spanning tree (which is cycle-free), together with a minimal set of non-contractible arc cycles, so called *generators* of the fundamental group $\pi_1(\mathcal{M})$ of $\mathcal{M}$.

Furthermore, there is no need to constrain nodes that do not lie on a critical entity, as other nodes are not related to the integer degrees of freedom. Also, constraining multiple nodes per critical entity is redundant due to alignment constraints in the original parametrization problem. Therefore, we compute a set of arc paths σ_i that form a spanning tree of the critical entities, augmented by a set of cyclic generator paths. This process is illustrated on an example in fig. 8.1.

8.2.1.1 Spanning Tree Construction

Note that the constraint (8.3) is three-dimensional, while not every coordinate of a critical entity must be integer. Assume a critical node n_0 exists; this node's vertex parameters must all be integers, which we could arbitrarily constrain to be $(0, 0, 0)$. With this fixed, any differential constraint of type (8.3) derived from an arc-path between n_0 and another node n_i would also force all three coordinates of n_i to integers. If n_i is another critical node, this is desired; for critical links and critical regions one or two coordinates must be dropped from the constraint, namely the coordinates that are not constant within the entity.

Assume now, that the spanning tree connects only critical nodes and one 3D constraint for each path from n_0 to n_i for $i \neq 0$. For each higher-dimensional critical entity incident on a critical node, this already fixes all degrees of freedom. If we connect additionally one arbitrary node per

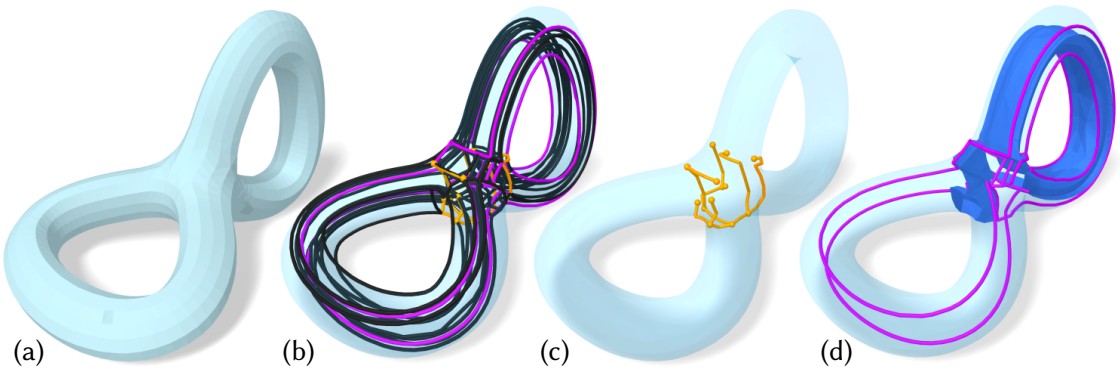


Figure 8.1: The process of determining arc paths to constrain is shown. (a) For a given input domain with (b) embedded T-mesh arcs, we determine (c) a minimal spanning tree (orange) of critical entities, and (d) a set of cycles (magenta) that generate the input's fundamental group. Cycles are determined by picking for each manifold piece of the domain's cut surface (dark blue) a single cycle that crosses the cut surface only once, through that piece. Note that, while this is not necessarily minimal, it is always sufficient.

cyclic critical link (which have no critical nodes), and constrain it with a 2D constraint, all critical nodes and links are fixed, and also any critical regions incident to either. This leaves only closed critical regions that are completely disjoint from other critical entities, which can be included into the tree by once more connecting one arbitrary contained node and constrain it in just a single dimension. Such a tree is minimal, as no constraint is redundant. It can be efficiently built via on the node-arc graph of the T-mesh $\mathcal{T}$: First, build any complete acyclic search graph G (e.g., breadth-first) of all nodes starting from n_0, consisting of nodes $n \in G$ and arcs $a \in G$. Then repeatedly trim leaf nodes n (and incident arcs a) from this graph, as long as n is neither a critical node, nor the last node of G on a critical link, nor the last node of G on a critical region.

For this we assumed a critical node n_0 exists that forms the root of G, which can be constrained to $(0, 0, 0)$. If no such node exists, the root node is instead chosen as the first existent node of: (i) any critical link node, partially constrained to $(0, 0)$; (ii) any critical region node, partially constrained to (0). The existence of (ii) is ensured by $\mathcal{T}$ having a boundary.

We have noted above that the constraint paths are rarely rooted at nodes that are not also original vertices. Using the procedure just explained, only the spanning tree nodes that are additionally chosen from cyclic critical links or closed critical regions can potentially represent non-original vertices; the rest, being critical nodes, must have existed in the input.

8.2.1.2 Cyclic Generators

The method of Kim et al. [2008] promises to exactly compute the cyclic generator loops of $\mathcal{T}$, or, more precisely, of its fundamental group $\pi_1(\mathcal{T})$. Their method is based on a series of tree-cotree-like decompositions [Eppstein 2003] on shrinking restrictions of an input tetrahedral mesh, called *k-skeletons* of $\mathcal{T}$. However, the explanation of the method in the original publication is vague and ambiguous; without an exact description of the algorithm, basing an implementation only on best-effort interpretations, it was easy to come up with counterexamples for which the *interpreted* algorithm would fail.

Instead we base our method of computing a (potentially non-minimal but sufficient) set of loops covering the generators of $\pi_1(\mathcal{T})$ on an extension of the method described in [Jiang et al. 2014, Fig. 5]. The original method aims to cut the domain into a topological ball, as part of deriving transition surfaces for integer-grid maps, called *cut surface* (cf. section 3.3). Intuitively, the cut surface is determined by initializing it as the full set of 2-cells, then agglomerating face-adjacent 3-cells and removing all traversed faces (2-cells) from the cut surface. The resulting

agglomeration of 3-cells, separated by the 2-cells of the cut surface, is of guaranteed ball topology, but the cut surface is not locally minimal; it has outgrowths that only partially penetrate into the ball-topology 3-cell agglomeration. These outgrowths can be iteratively shortened.

In the current setting, the cut surface does not need to account for singularities or transitions; we only care about the overall domain, not the map. On $\mathcal{T}$, computing it algorithmically amounts to the following:

(1) Initialize the cut surface $\mathcal{S}$ as the set of all patches P.

(2) Starting from any block $b \in B$, grow a dual search tree (with blocks as graph vertices and patches between block pairs as graph edges) until all blocks are conquered.

(3) Remove all patches traversed by the tree, as well as all boundary patches $p \in \partial\mathcal{T}$, from $\mathcal{S}$.

(4) Iteratively trim 'open ends' of $\mathcal{S}$, i.e., remove all patches p that have an arc $a\partial p$ incident to no other patch of $\mathcal{S}$.

The resulting $\mathcal{S}$ potentially consists of multiple connected components, each of which is not necessarily 2-manifold. Cutting it into multiple manifold pieces at all non-manifold points, results in pieces that must be genus 0 (with boundary); otherwise they would not cut $\mathcal{T}$ into a topological ball, which would contradict the construction.

One can identify three types of loops on $\mathcal{T}$ based on their interaction with $\mathcal{S}$:

(A) Any loop that is unbroken by $\mathcal{S}$ is topologically trivial, i.e., contractible.

(B) Any loop on $\mathcal{T}$ that is split into exactly a single curve by $\mathcal{S}$ is topologically non-trivial, i.e., incontractible.

(C) Loops that are broken into more than one curve can be either; some parity disambiguation would be required.

However, we can recognize that any topologically non-trivial loop classes of type (C) can be composed by concatenation of type (B) loop classes. Hence, (B) must contain at least a set of generators of $\pi_1(\mathcal{M})$. Note also that it makes no difference for path homotopy where *exactly* a path pierces a given manifold piece s of $\mathcal{S}$. Thus, we can conclude that a single loop per manifold piece s of $\mathcal{S}$—crossing $\mathcal{S}$ only once, through s—covers a generating set of $\pi_1(\mathcal{T})$.

This choice is not perfectly minimal (cf. fig. 8.1), due to non-manifoldness potentially resulting in more pieces of $\mathcal{S}$ than strictly necessary, but this causes only constraint redundancy (not overly strict or infeasible constraints). Loops per cut surface piece s are easily determined by choosing any node n_s on s, and employing a shortest path search on the arc-node-graph that starts at n_s, expands only via one side of s, does not cross $\mathcal{S}$ and ends when reaching n_s from the other side of s. The overhead of doing this a (moderately) non-minimal number of times is also negligible.

8.2.2 Obstacles

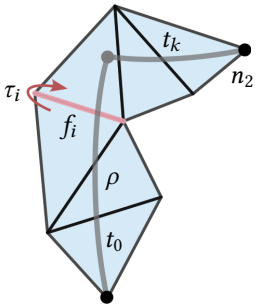
Translating integer arc paths to global constraints on vertex parameters of the original background mesh is non-trivial for two main reasons, further explained in the following: The presence of non-trivial transitions along the paths, and the connectivity mismatch between *original* tet mesh and the *refined* tet mesh in which the T-mesh is embedded (section 5.6.2.1).

Transition-aware constraints In a transition-free setting, we could just "integrate" the quantization values within a simply-connected global chart of $\mathcal{T}$ (self-adjacent via transitions across patches), and directly prescribe integer vertex parameters. However, the obtained integers would only make sense for this particular choice of chart layout, and imposing the same on the tetrahedral mesh $\mathcal{M}$ for the final parametrization can be challenging because interior facets of $\mathcal{T}$ do not necessarily coincide with (original) facets of $\mathcal{M}$ or because the chart layout of the seamless parametrization from the first step is to be adopted (e.g., because a guiding frame field is represented in it). The challenges outlined above can be circumvented by constraining parametric differences of vertex parameters rather than absolute integer values, in a manner that takes transitions into account, which consist of fixed (signed) coordinate permutations and translation variables.

Background domain mismatch To formulate constraints on original vertex parameters, we specifically need to consider the background tet mesh $\mathcal{M}$ in the state fresh after computation ϕ. This is the setting in which fragile rounding approaches have formerly been employed in; to function as a drop-in replacement our method's constraints must likewise apply to the variables of this original problem. Concretely, this means that we need to consider a tet mesh connectivity in which the T-mesh $\mathcal{T}$ would not necessarily be embedded into discrete elements; recall, that to embed $\mathcal{T}$ into the background, a refinement during T-mesh construction was required (section 5.6.2.1). Because the only connectivity operations performed were splits—introducing new cells and splitting existing ones into smaller parts (the union of which still represents the same region)—we can store a history of splits and interpret the original $\mathcal{M}$ as a subset contained within the finer version on which the quantized T-mesh lives. Essentially, we can blank out any cells that are truly 'new' and not a subset of an original one for the purpose of constraint extraction, while still using them for local mesh navigation. In the following we abstract from this connectivity correspondence, simply modeling it as a geometric overlay of the T-mesh $\mathcal{T}$ on top of the original tet mesh $\mathcal{M}$ in object space. A backup of the original (sanitized) parametrization ϕ and implied original transitions τ are likewise assumed to be readily available.

8.2.3 From Arc-Paths to Constraints

For a directed arc path ρ, starting at node n_1 and ending at node n_2, let $T_\rho = (t_0, \dots, t_k)$ denote a sequence of facet-adjacent tetrahedra of $\mathcal{M}$ such that it contains ρ, t_0 contains n_1, and t_k contains n_2—as illustrated on the right in a 2D sketch. Let $F_\rho = \{f_1, \dots, f_k\}$ be the sequence of directed facets in between, where f_i is the facet carrying the transition τ_i from t_{i-1} into t_i. Let $\tau_{i\to k} = \tau_k \circ \cdots \circ \tau_{i+1}$ denote the composed transition from tet t_i into the last tet t_k. Then $\tau_\rho = \tau_{0\to k}$ denotes the composed transition along the entire path ρ. The direction $\vec{a}|_{t_k}$ of any arc $a \in \rho$ in the coordinate system of t_k can be determined as follows: Let $n_a \in \partial a$ be the arc's endpoint closer to n_2 along ρ, $t_i \in T_\rho$ be the tet that contains n_a, and $\vec{a}|_{t_i}$ be the direction of a pointing towards n_a in t_i (which can be read off a's embeddding in t_i). Then $\vec{a}|_{t_k} = \tau_{i\to k}\vec{a}|_{t_i}$, where we apply only the (fixed) rotational part of τ because we transform a unit direction vector, not a coordinate.



Let $\Delta_q \boldsymbol{u} \in \mathbb{Z}^3$ be the vectorial sum of arc vectors of ρ expressed in t_k, i.e., $\Delta_q \boldsymbol{u} = \sum_{a\in\rho} q_a \vec{a}|_{t_k}$. Assume nodes n_1, n_2 lie on vertices v_1, v_2 of $\mathcal{M}$ and the relevant sector parameters are $\boldsymbol{u}_{v_1}$ and $\boldsymbol{u}_{v_2}$ (within the sectors around v_1 and v_2 traversed by ρ). Using the following condition, we can constrain the parametrization to have the spacing between v_1 and v_2 that is intended by q:

$$\boldsymbol{u}_{v_2} - \tau_\rho \boldsymbol{u}_{v_1} = \Delta_q \boldsymbol{u}. \tag{8.3}$$

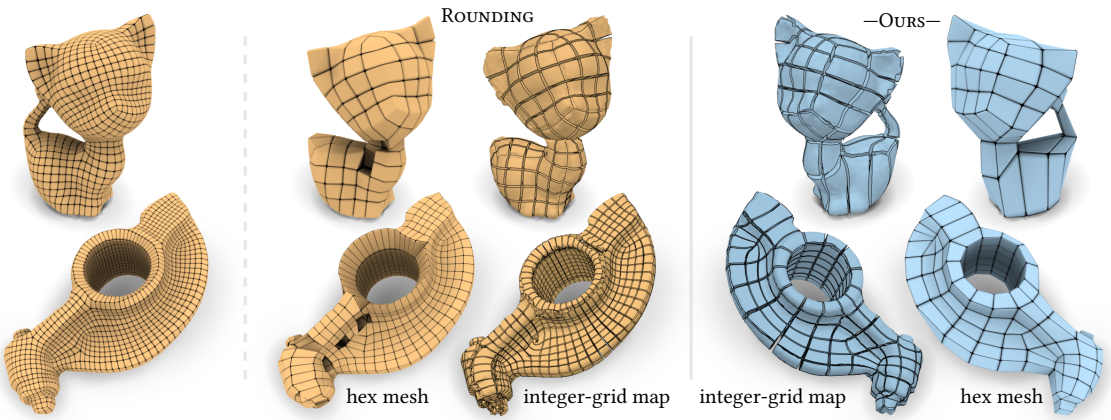


Figure 8.2: Volumetric integer-grid maps for hex meshing are commonly generated using a classical integer rounding procedure (orange). This rounding is fragile, its probability of success depends in particular on the desired grid resolution. While for sufficiently fine results it can be successful (far left), for a coarser target resolution, map degeneracies emerge (center right), implying defects in the mesh (center left); notice the holes, the missing tail, and an entire missing outermost layer, exposing interior singularities. Our method (blue), by contrast, is guaranteed to yield conflict-free integer values regardless of resolution, even if extremely coarse. Besides the meshes, the underlying integer-grid maps are visualized via cut out integer-iso-surfaces on the original model.

Both $\boldsymbol{u}_v$ represent three parameter variables each. Note that also the transitions $\tau = (R, \boldsymbol{d})$ are not constants: Transition rotations R are fixed, but (integer) translations $\boldsymbol{d}$ are variable, and each coordinate of $\boldsymbol{d}_\rho$ is a sum over one or more such variables. In case a node does not lie on a vertex, we instead use a linear combination of up to four vertices (of a containing tet, using barycentric coordinates) to express the constraint—though this is rarely necessary, as explained below.

Constrained Parametrization We now can add these constraints to the volumetric seamless parametrization problem. They pin down exactly the integer degrees of freedom, so the resulting parametrization will be an integer-grid map even when optimizing over the continuous domain. Next, we show, analyze, and discuss integer-grid maps and hex meshes generated in the here described way.

8.3 Results: Integer-Grid Maps and Hex Meshes

We base our evaluation on the T-mesh quantizations obtained in section 7.6. The constraint extraction routine is included in the QGP3D library discussed in section 7.5.1. The hex meshes we show are extracted from the generated integer-grid maps using HexEx [Lyon et al. 2016].

Let us first compare to the rounding-based approach. We apply the best existing variant, incremental rounding with interleaved optimization [Li et al. 2012; Jiang et al. 2014], to the dataset of seamless maps. Note that since the problem structure (in terms of the singularity network, boundary alignment) is identical, the only difference is the integer determination strategy.

The only known way to verify the validity of an integer assignment generated by rounding is via a valid (locally injective) map that adheres to it and thereby certifies its validity [Bommes et al. 2013]. Given the remaining limitations of existing methods for 3D map computation (cf. section 3.6.2), this can only conservatively be checked. For a fair comparison, for the purpose of this experiment we also consider the quantizations generated by our method valid only if a valid map for them is found using the same map computation setup. We use the optimization setup of [Jiang et al. 2014, Sec. 6] including adaptive stiffening (raising the iteration limit from 5

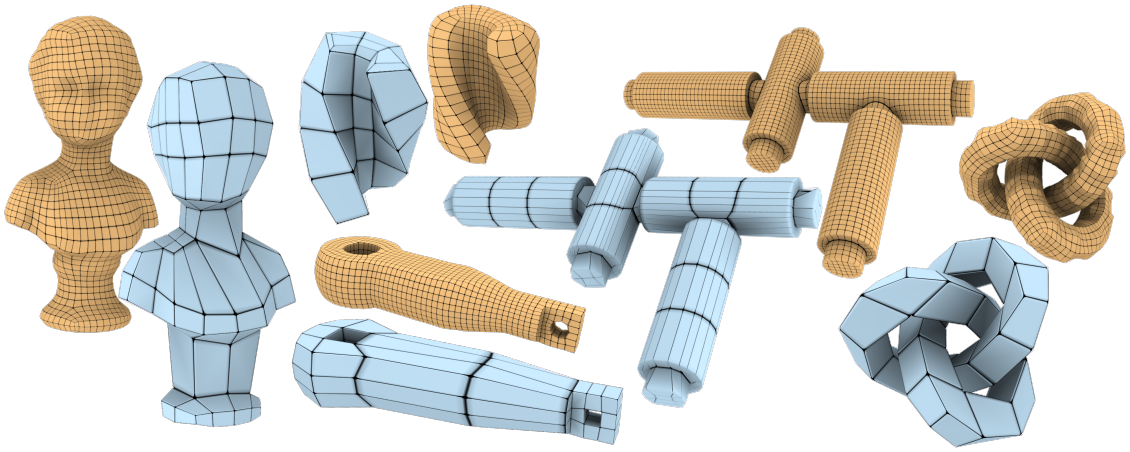


Figure 8.3: Orange: Coarsest hex meshes for various models achievable using rounding. Any coarser target resolution leads to severely invalid integer-grid maps and defective hex meshes. Blue: Some much coarser hex meshes generated using our quantization method.

Table 8.1: For inputs from our two datasets, the number of hexes in the simplest mesh that is achieved with either rounding or our method (as well as their ratio) is shown. Furthermore, the columns SJ_r and SJ_q show a comparison of the mean scaled Jacobian quality between then simplest mesh achieved with rounding and a mesh of similar resolution generated using our quantization method. (The complete lower table was published in the appendix of [Brückler et al. 2022a]).

Model	SJ_r	SJ_q	round	**ours**	$\frac{\text{ours}}{\text{round}}$
KITTEN	0.90	0.90	3276	139	4%
SCULPTURE	0.93	0.93	386	18	5%
ROCKERARM	0.91	0.86	8056	571	7%
CYLINDER	0.78	0.78	60	5	8%
JOINT	0.95	0.95	715	62	9%
BONE	0.80	0.80	628	70	11%
SPHERE	0.71	0.71	32	7	22%
FANDISK	0.84	0.85	162	51	31%
PRISMA	0.85	0.85	9	3	33%
CAMILLE HAND	0.81	0.79	1438	773	54%
ARMADILLO	0.92	0.92	29306	16393	56%
BROKEN BULLET	0.80	0.80	36	24	67%
CUBE SPHERE	0.85	0.85	10	10	100%
FANPART	0.92	0.92	5	5	100%
TETRAHEDRON	0.81	0.81	4	4	100%

Model	SJ_r	SJ_q	round	**ours**	$\frac{\text{ours}}{\text{round}}$
KNOT-HEX	0.84	0.84	4704	45	1%
CHAMFER L4	0.95	0.96	484	7	1%
TWISTEDU	0.81	0.80	1227	19	2%
EXAMPLE 2	0.92	0.93	30334	695	2%
ROTELLIPSE PADDED	0.78	0.78	804	19	2%
EXAMPLE 5	0.99	0.99	37	1	3%
DOUBLE HINGE WH	0.95	0.94	2592	81	3%
BU HEX OPT	0.89	0.87	8337	300	4%
TABLE2 POLYCUBE OUT	1.00	1.00	340	13	4%
CUBESPIKES MODEL IN	0.95	0.96	5895	241	4%
EXAMPLE 4	0.88	0.92	15799	660	4%
GEAR	0.95	0.95	1456	69	5%
FERTILITY HEX-LARGEL	0.89	0.89	7693	379	5%
KITTY	0.91	0.91	2522	137	5%
HOLLOW-EIGHT-HEX	0.89	0.89	6757	384	6%
⋮					

to 30 to increase chances even further). The difference in robustness of the two approaches is demonstrated in fig. 8.2 on two representative examples.

Coarseness Using a binary search over the target resolution parameter, for both approaches (rounding and MC-based quantization) we determine the coarsest resolution for which they yield an integer assignment that is valid in the sense that a locally injective integer-grid map is achieved using the above setup. The results are reported in table 8.1, and fig. 8.3 shows some examples. The fragility of rounding becomes obvious: In some cases the simplest valid map obtainable with rounding is around 100× more complex than that obtainable based on our method.

Quality As a further comparison, beyond validity, let us consider the relative quality of results, in terms of mesh quality measured in meshes extracted from the generated integer-grid maps. Figure 8.4 shows plots of the mean scaled Jacobian quality measure, over output meshes of varying resolution. It can be observed that, for finer resolutions on which rounding also succeeds, our method does not differ much, i.e., its reliability does not come at a significant cost of quality in

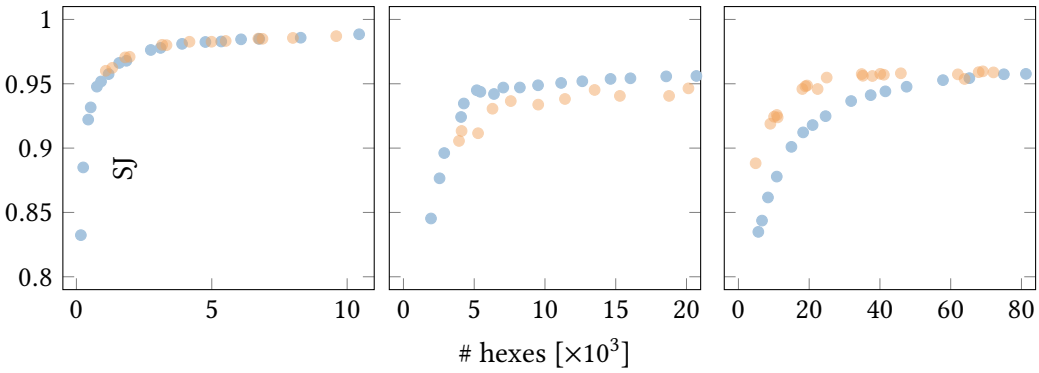


Figure 8.4: Comparison between our method (blue) and rounding (orange) in terms of resulting hex mesh quality (mean scaled Jacobian) for varying output mesh resolution. Left: typical/average case (model JOINT). Center and right: best and worst cases (EXAMPLE2 and ROCKERARM) over the entire dataset. Also see the SJ columns in table 8.1.

general. Figure 8.6 shows some visual examples. Another quality comparison (regarding the employed quantization value ranges, $\mathbb{Z}^{>0}$ or $\mathbb{Z}$) is shown in fig. 8.5.

8.3.1 Hex Mesh Input

Finally, let us just briefly point out another potential use. As in previous chapters, hex meshes can also be taken as input—they trivially induce a seamless parametrization. We can then generate an output hex mesh with identical singularity structure, but different resolution or different *base complex*. The base complex is the coarsest conforming partition of the mesh into regular pieces, i.e., blocks of some numbers $m \times n \times o$ of hexes. A mesh with sufficiently simple base complex is said to be block-structured, and is desirable in certain contexts (cf. section 2.3.3.2).

To improve (i.e., simplify) the base complex, take a hex mesh as input, compute a coarsely quantized integer-grid map, and scale it by an integer factor, aiming to match the input mesh resolution (or a possibly deviating user-defined resolution). To illustrate the potential, for a selection of hexahedral meshes from the repository of [Bracci et al. 2019] we report the number of blocks in the base complex before and after this processing in table 8.2. Depending on the use case, of course further control over the base complex' quality beyond just simplicity may be needed, e.g, as explored for the 2D case by [Lyon et al. 2021a].

Let us note that [Gao et al. 2017b] also consider a base complex simplification problem. They take a more powerful approach that allows the singularity structure to be modified, granting more flexibility. Assuming a scenario where the singularity structure is to be preserved, our approach may offer some advantage: the non-conforming structure of the MC our method builds on, allows for a broader range of (singularity-preserving) modifications than the conforming sheet collapses used in the above method. The relevance to the specific task at hand appears to be not huge, though; while we observed a further reduction relative to sheet collapses by a factor of 5× in one case, in most other cases the factor is lower than 2×.

8.4 Discussion

Injectivity Let us remind that the general problem of valid (locally injective) volumetric parametrization still lacks a fully robust solution (section 3.6.2), so also this particular optimization formulation is not guaranteed to yield a locally injective map (whether with or without the integer constraints). For orientation: for the most challenging, maximally coarse quantizations considered

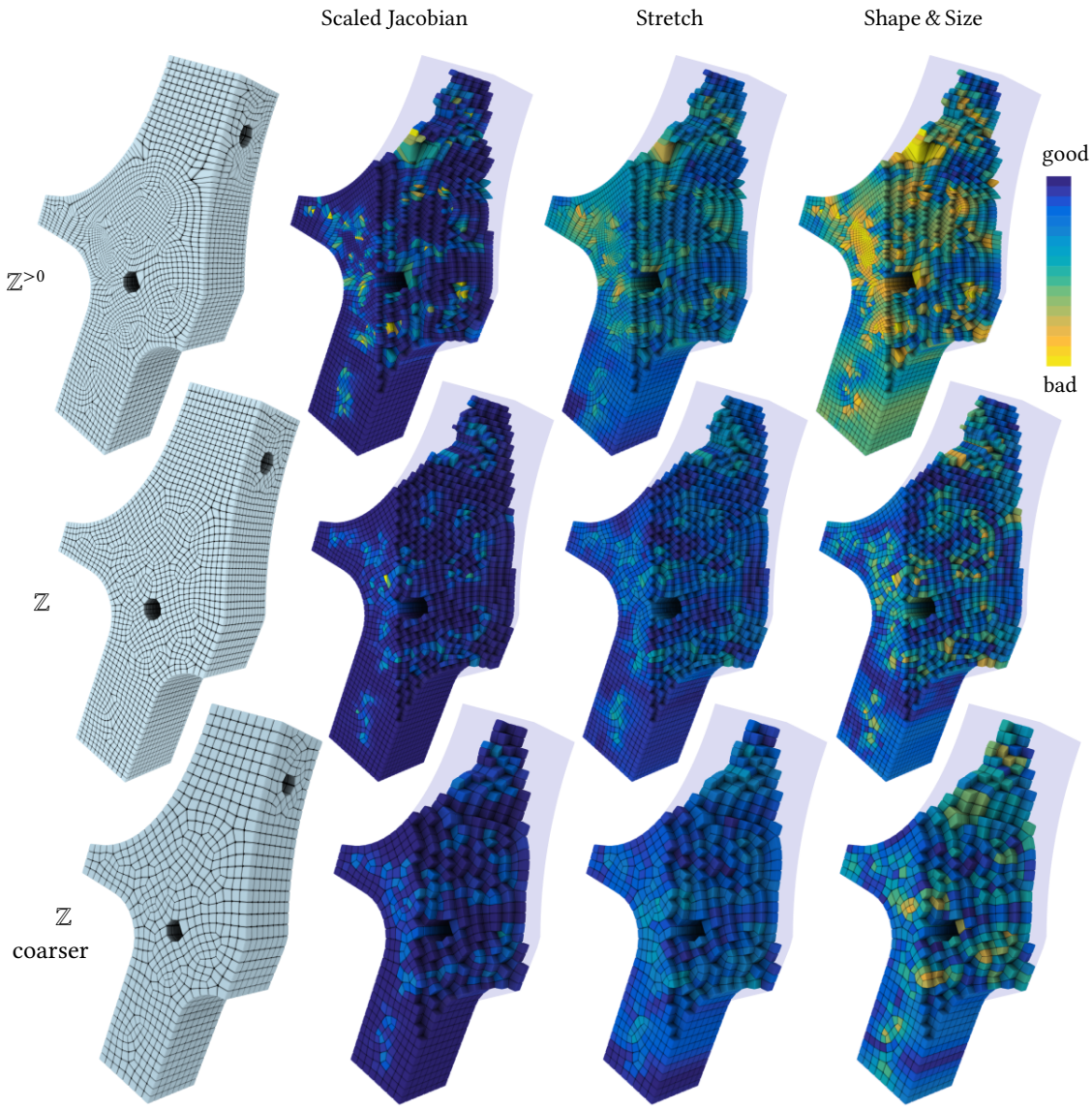


Figure 8.5: Top: Coarsest hex mesh that can be achieved using $q_a \in \mathbb{Z}^{>0}$, together with visualization of three quality measures [Stimpson et al. 2007]. Middle: A mesh of equal resolution generated using $q_a \in \mathbb{Z}$. Its quality is significantly better. Bottom: Using this larger value range, also coarser meshes can be achieved, still with higher quality than in the finer top row.

in the experiments in section 7.6.2, reparametrization yields a fully valid map in 84% of the cases. With the integer constraints determined by the proposed method, it is clear that an occasional failure is due to this general shortcoming, not due to infeasible constraints. In fact, in the next chapter we demonstrate a way to initialize the map reparametrization with a guaranteed-injective starting point that adheres to these constraints. This demonstrates constraint feasibility and closes the final robustness gap, ensuring a theoretical 100% success rate of obtaining a valid integer-grid map from a seamless map.

Hex Geometry Recall also that while a valid integer-grid map always implies a *structurally* valid hex mesh, the question of *geometric* validity is more intricate. The map associates each logical hex element with some unique non-degenerate region of the input object, but in practice often geometrically simple (e.g., trilinear) elements are assumed. Map injectivity guarantees geometrically valid trilinear hexes only for a sufficiently fine sampling (section 3.6.2). For instance, at the maximally coarse resolutions in 46% of the implied meshes all (trilinear) hex elements

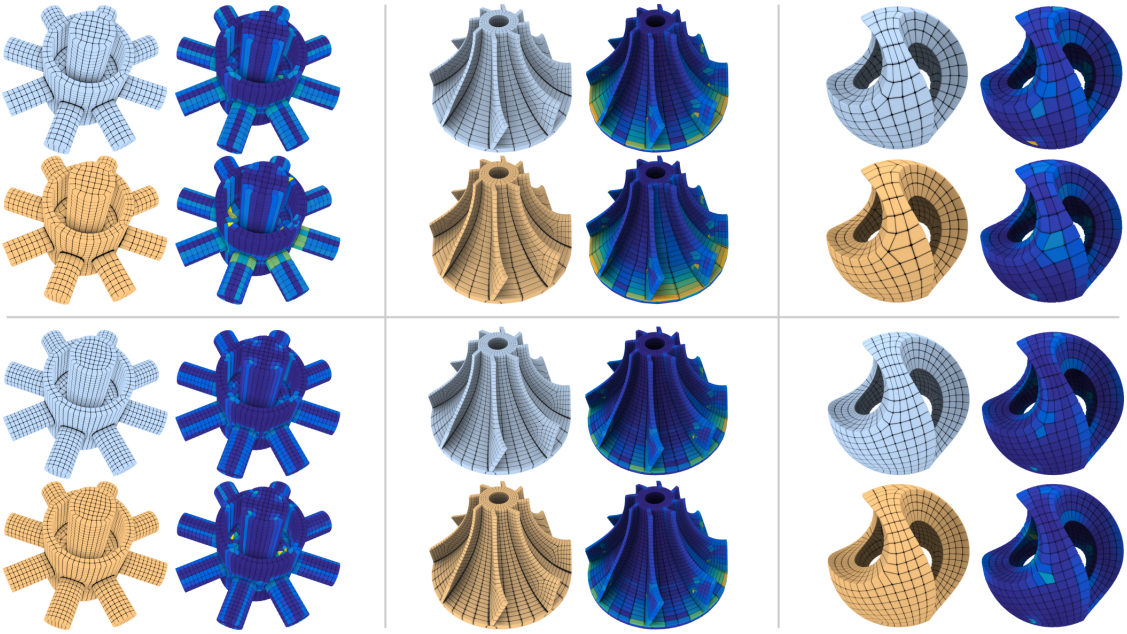


Figure 8.6: Hex meshes of three example models, in a coarser (top) and finer (bottom) version. Integers were determined using our method (light blue) or rounding (orange). Next to each mesh, a visualization of per-element scaled Jacobian values is shown. Color scale as in fig. 8.5. Notice that rounding behavior is not monotonic: In the bottom right case, despite being of higher resolution than the top case, the integer-grid map generated by rounding had defects; HexEx managed to extract a mesh anyway, albeit at the cost of disrespecting the input singularity structure (singularity merged into boundary in the front).

Table 8.2: Improvement of block-structuredness (base complex simplicity). Reported is the number of base complex blocks in the input (BC in), the number achieved using sheet collapses (BC sheet), the number achieved using our method (BC ours), as well as ratios of them.

Model	**BC in**	BC sheet	**BC ours**	$\frac{\text{ours}}{\text{sheet}}$	$\frac{\textbf{ours}}{\textbf{in}}$
EXAMPLE 1	74331	59691	11065	19%	15%
ARMADILLO HEX-B	3265	640	505	79%	15%
BUNNY HEX	1282	300	221	74%	17%
ARMADILLO HEX-A	5960	1263	1031	82%	17%
GARGOYLE	7563	2102	1419	68%	19%
BUNNY HEX	1324	254	258	102%	19%
PEGASUS-HEX	9745	3945	2032	52%	21%
DRAGON-HEX	12488	3299	2770	84%	22%
DANCING-CHILDREN-2	5482	1656	1313	79%	24%
IMPELLER	878	326	224	69%	26%
BUNNY MODEL IN	637	260	165	63%	26%
ELEPHANT HEX	3105	970	835	86%	27%
FERTILITY	1352	934	375	40%	28%
BUSTE HEX	1081	362	303	84%	28%

have positive scaled Jacobian values. With increasing resolution, this rate increases, e.g., at 1.5× higher quantization target resolution to 76%, at 2.0× to 82%. Note that this aspect is common to parametrization-based methods in general, not particular to our quantization approach, and deserves further attention in future work.

T-mesh Collapsing for Guaranteed-Injective Maps

9

IV Quantize the seamless map ϕ into an integer-grid map ϕ_q.

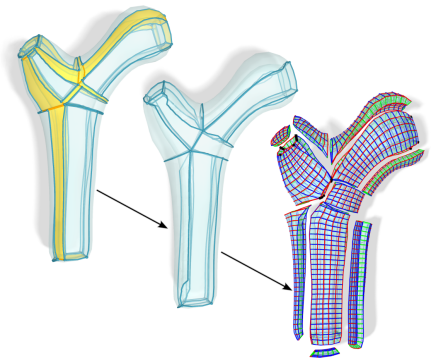
IVa–c A quantized, ϕ-aligned T-mesh $\mathcal{T}_q$ in the volume

IVd Get constraints from $\mathcal{T}_q$ that force $\phi \mapsto \phi_q$

IVe Collapse zero-cells from $\mathcal{T}_q$

IVf Initialize ϕ_q per cell, globally optimize it s.t. IVd

V Extract a hex mesh from ϕ_q.



In the previous chapters, we have established how to compute integer parametric dimensions for the cells of a T-mesh, and how to use these to constrain a global computation of an integer-grid map, such that an injective map adhering to these constraints can be found via optimization over a purely continuous domain. While the constraints are guaranteed feasible by construction, the optimization process itself is still fragile, and may fail to find a valid map even though one exists. This is the final robustness gap that remains to be solved in this chapter.

The difficulty in map optimization arises primarily from the potential complexity of input and target domain, together with the imposed injectivity and seamlessness constraints. Injectivity constraints, being highly non-convex, are particularly challenging to satisfy for complex domains. For simpler parameter domains, specifically ones of ball-topology and convex shape, reliable methods to compute injective maps exist.

This calls for a *union-of-maps* approach, decomposing the complex mapping problem into multiple simpler ones, each of which can be solved reliably. The T-meshes established in previous chapters serve as the decomposition, their quantization as a prescription of an integer parametric cuboid per block, agreeing across shared interfaces. However, as also established, these T-meshes may contain undesirable elements, in particular arcs, patches and blocks assigned a target extent of zero. A naive per-block map would then necessarily be degenerate (non-injective) on these elements. Our solution to this problem is to eliminate such undesirable elements via collapse operations, a common operation for standard polyhedral meshes (cf. section 2.2.2). In the most common variant of polyhedral meshes, based on linear elements, the geometry of each cell is inductively defined just from vertex coordinates, hence geometry updates are already implied by connectivity changes. In case of a coarse T-mesh embedded into the connectivity of a background tet mesh, it is not immediately clear how the geometry, defined through the embedding, must be updated.

The core problem is therefore, to find sequences of connectivity updates that eliminate

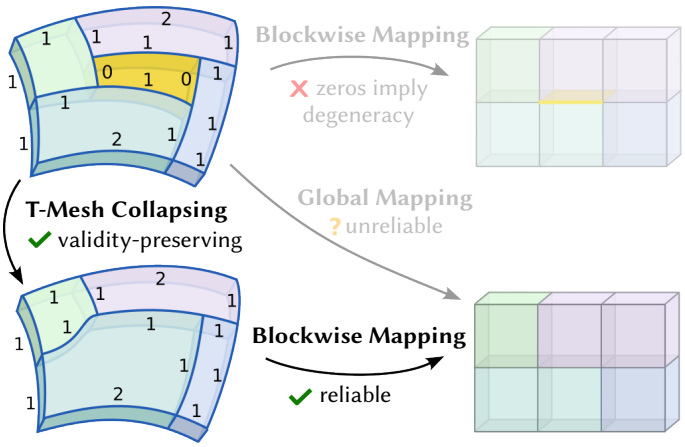


Figure 9.1: Top: The complex contains a *zero-block*; mapping this block (which has non-zero volume in the object) onto an accordingly sized cuboid (of zero volume) forces the map to be non-injective. Bottom: After performing collapses of zero-elements, maintaining a bijective embedding of the remaining complex, the blocks can be mapped without degeneration. Diagonally: Computing the map not blockwise but *globally*, is unreliable with current methods.

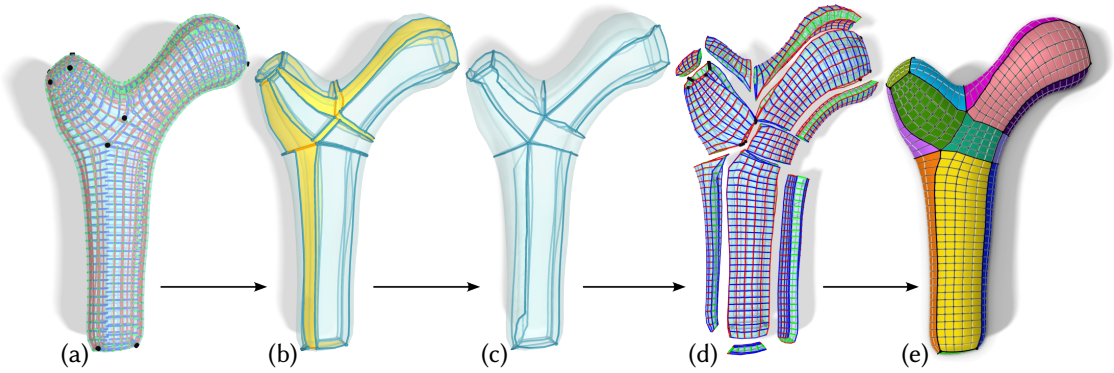


Figure 9.2: (a) A *seamless parametrization* in the volume of an object. (b) A T-mesh embedded within the object and aligned with this parametrization; some of its elements (marked yellow, orange) are determined in a *quantization* process to be undesirable. (c) The result of eliminating all marked elements using our proposed collapsing strategy. (d) An exploded view of an *integer-grid map*, safely constructed block-by-block based on this simplified cell complex, and optimized for reduced distortion. (e) A hex mesh extracted from this map, exhibiting a conforming multi-block structure.

unwanted elements, while also updating the embedding *compatibly*. In the following we present a set of operators to perform modifications, in particular collapses and splits, in generic volumetric cell complexes which are discretely embedded in a background mesh. Topological integrity and embedding validity are carefully maintained. We apply these operators strategically to T-meshes to obtain a simplified T-mesh, readily usable for robust block-wise mapping.

9.1 Related work

Volumetric mapping The computation of proper (e.g., bijective) maps between volumetric domains is a challenging problem. For simple maps (without singularities, cuts, overlaps, etc., as in IGMs) some recent progress can be observed: A constructive approach offering guarantees regarding bijectivity of the resulting map [Campen et al. 2016; Hinderink and Campen 2023; Hinderink et al. 2024], a recent progressive approach [Nigolian et al. 2023; Nigolian et al. 2024], or optimization based techniques that achieve high success rates [Du et al. 2020; Garanzha et al. 2021]. For more general maps (IGMs and seamless maps) with prescribed singularities, cuts, chart transitions, and pin constraints, only best-effort techniques (based on non-convex numerical optimization) are available [Nieser et al. 2011; Pietroni et al. 2022], with an increasingly high level of fragility the more constraints are imposed. Even in the significantly simpler analogous 2D setting, while significant progress in terms of reliable computation can be witnessed [Zhou

et al. 2020; Levi 2022; Shen et al. 2022], no fully satisfactory solution treating the problem as a whole (without decomposition) in full generality is in sight so far. This leaves little hope that a reliable solution for the *direct* computation of valid 3D IGMs will be found soon. This is our key motivation to instead reduce this problem, which appears in the parametrization based hexahedral meshing pipeline, to multiple instances of a simpler mapping problem.

Component-based mapping Reliable algorithmic solutions with strict guarantees typically focus on restricted settings, such as mapping onto flat domains with convex boundary. To handle more general settings, a strategy that has been followed in a variety of ways is to decompose the more general problem into multiple instances of a simpler special case problem. In other words: *Build the desired map out of multiple simpler maps.* This can be done either through composition or through union of maps. Examples include:

- Maps from surfaces to non-convex planar domains, composed out of two maps via a convex planar domain [Weber and Zorin 2014].
- Maps between surfaces of disc topology, composed out of two maps via a convex planar domain [Kanai et al. 1997; Schmidt et al. 2019].
- Maps to non-convex planar domains, as a union of multiple maps to convex planar domains [Kraevoy et al. 2003; Myles et al. 2014; Lyon et al. 2019, 2021b].
- Maps between surfaces of arbitrary topology, as a union of compositions of maps via convex planar domains [Kraevoy and Sheffer 2004; Schreiner et al. 2004; Schmidt et al. 2020].
- Maps between ball-topology volumes of arbitrary shape, as a union of ball-topology maps to star-shaped domains [Hinderink et al. 2024].

In one case [Lyon et al. 2019] a surface T-mesh is used to decompose a global mapping problem into simpler rectangle-mapping problems. All the above examples concern the case of 2D maps, though. We follow a similar route in 3D, to obtain a global, injective volumetric integer-grid map from simpler convex cube-mapping problems, with a reliable solution available.

Mesh Operators In many of the above works operators to (often incrementally) construct such meta meshes together with an embedding are described, specialized to the concrete use case. Operators to modify such meshes after the fact are rarely discussed. While modification operators (such as collapses, splits, and flips) of simplicial (triangular and tetrahedral) meshes (and to a lesser extent more general polygonal/polyhedral meshes) purely in terms of connectivity are well-explored [Freitag and Ollivier-Gooch 1997; Trotts et al. 1998; Botsch and Kobbelt 2004; Daniels et al. 2009; Peng et al. 2011; Gao et al. 2015a; Shen et al. 2021], embedded meta meshes require an accompanying update of the discrete embedding, so as to keep it consistent with the connectivity. The work of Lyon et al. [2019] touches the aspect of embedding collapse of 2D T-meshes only very briefly and provides few details on its concrete implementation. We are not aware of detailed and sufficiently general treatments of this aspect in the literature, in particular not for the non-simplicial and volumetric case, as required for our problem setting.

9.2 Notation

We use in the following the notation established in section 6.2, summarized here for convenience.

Background and meta-mesh The background mesh, defining the geometry in which the another cell complex is embedded, is a tetrahedral mesh $\mathcal{M} = T \cup F \cup E \cup V$, consisting of tetrahedra $T = \{t_1, t_2, ...\}$, facets $F = \{f_1, f_2, ...\}$, edges $E = \{e_1, e_2, ...\}$, and vertices $V = \{v_1, v_2, ...\}$.

An abstract meta-mesh is defined as $\mathcal{T} = B \cup P \cup A \cup N$, consisting of blocks $B = \{b_1, b_2, ...\}$, patches $P = \{p_1, p_2, ...\}$, arcs $A = \{a_1, a_2, ...\}$, and nodes $N = \{n_1, n_2, ...\}$. Both are equipped with cell-wise incidence relations ∂ and ∂^{-1}.

Meta-mesh cells In the setting discussed here, $\mathcal{T}$ is a T-mesh as defined in theorem 6.3. These are of particular interest for hex meshing, as blocks are structurally cubes, consisting of six sides and eight corners. The more restricted case of conforming cuboidal blocks, is a special case of T-meshes, where each cube side consists of exactly one patch, and each cube edge of exactly one arc. The majority of operators we present, however, are applicable to more general volumetric cell complexes as well. At the very least, though, we assume that n-cells of the abstract meshes represent open n-balls and that incidences are restricted accordingly. Note that this does not imply that a cell, together with its boundary, is homeomorphic to a closed n-ball, as self-adjacencies may exist; a block may be linked to itself via one of the cells from its boundary. Such configurations are relevant for some applications, and may also occur in intermediate states when performing structural modifications (like the collapses in our use case).

Geometry and Embedding In our setting, linear elements for $\mathcal{M}$ are assumed, hence the geometry of the background mesh $\mathcal{M}$ is given as per-vertex coordinates—one per vertex for object space coordinates, and one per vertex for each parametric chart covering the vertex. The geometry of a cell of $\mathcal{T}$ is defined via that of the elements of $\mathcal{M}$ it is embedded in: A discrete embedding $\mathcal{I} : \mathcal{T} \hookrightarrow 2^{\mathcal{M}} \setminus \varnothing$ maps each cell of $\mathcal{T}$ onto a set of elements of $\mathcal{M}$, respecting conditions ($\mathcal{I}_\text{I}$) to ($\mathcal{I}_\text{III}$), as detailed in section 6.2.4.

Concretely, the embedding must be at least injective, and structure-preserving: No two meta-cells of $\mathcal{T}$ may overlap, and the incidences of meta-cell embeddings within $\mathcal{M}$ must topologically match abstract cell incidences within $\mathcal{T}$. Optionally, surjectivity can be required, meaning that all elements of $\mathcal{M}$ are covered by the embedding of some meta-cell of $\mathcal{T}$. Crucially, all operations that modify $\mathcal{I}$ must maintain these conditions.

Notation 9.1 (Backwards Embedding). Disjointness allows us to define a kind of inverse: We write $\mathcal{I}^{-1}(x) = c$ if the background cell $x \in \mathcal{M}$ is part of the embedding of the meta-cell $c \in \mathcal{T}$, and $\mathcal{I}^{-1}(x) = 0$ if it is part of no such embedding. We call this $\mathcal{I}^{-1}$ the *backwards* embedding. For a set C of meta cells, $\mathcal{I}(C) \coloneqq \bigcup_{c \in C} \mathcal{I}(c)$ is the union over cell-wise embeddings, analogously for a set X of background cells $\mathcal{I}^{-1}(X) \coloneqq \bigcup_{x \in X} \{\mathcal{I}^{-1}(x)\}$.

For implementation purposes, it is convenient to keep track of both $\mathcal{I}$ and $\mathcal{I}^{-1}$ explicitly to allow for quick access in either direction.

T-mesh Quantization In the concrete use case considered here, the meta mesh is a T-mesh enhanced by a quantization q represented as a non-negative integer length $q_a \geq 0$ per arc a. Negative arc lengths can not be realized by local collapse operations; because they provide only minute advantages (cf. table 7.1) precluding them does not significantly shrink the result space. Due to some arcs having zero-length, the quantization implies a metric, under which certain elements coincide that are separate in terms of T-mesh connectivity. Within a given block of the T-mesh, a local coordinate system can be defined based on arc directions and lengths. This local q-chart implies integer coordinates for each node, as well as integer-sized intervals, rectangles or cuboids for arcs, patches or blocks. Any set of points within a block that share the same implied integer coordinates coincide in the q-metric, and hence in the target integer-grid map. These coincidences transitively extend across block boundaries. Any coincidence of separate elements

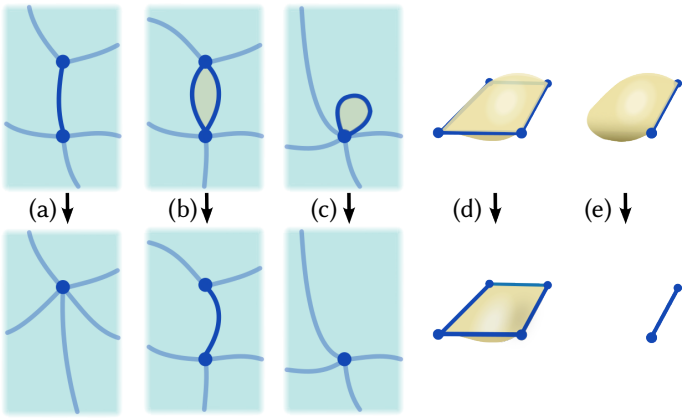


Figure 9.3: Illustration of basic collapse types. (a) collapse of an arc with two nodes. (b) collapse of a pillow patch with two arcs. (c) collapse of a pillow patch with one arc; this is equivalent to the contraction of the surrounding cyclic arc to a point. (d) collapse of a pillow block with two patches. (e) collapse of a pillow block with one patch; note this is equivalent to the contraction of the surrounding patch to a curve.

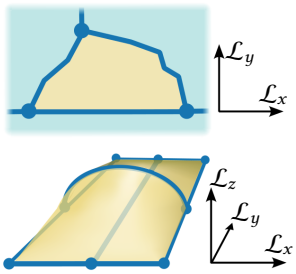
implies integer-grid map degeneracy, unless these separate elements are made to coincide in the T-mesh itself. Thus, the coincidence relations derived from q serve as instructions guiding the collapsing of T-mesh elements.

9.3 Collapsing General Embedded Cell Complexes

The main operators we require are collapse operators, for arcs, patches, and blocks of a volumetric cell complex. Their purpose is to remove the respective cell from the complex while updating the surrounding cells to preserve all desired properties. Concretely, we start by describing the following atomic operators:

- collapsing a single arc, bounded by at most two nodes;
- collapsing a single patch, bounded by at most two arcs;
- collapsing a single block, bounded by at most two patches.

While an arc is naturally incident to no more than two nodes (one in the case of a loop arc), patches and blocks in a generic cell complex may be incident to arbitrary higher numbers of arcs and patches, respectively. For these a collapse is, in general, not uniquely defined in terms of connectivity. We therefore focus on the above simple types, called *pillow patches* and *pillow blocks*, first of all. Figure 9.3 shows a geometric illustration arc, pillow patch and pillow block collapses.



Uninformed collapsing of T-meshes does not necessarily yield only simple pillow blocks and patches; for instance, collapsing a patch along one dimension may leave more than two bounding arcs as shown in the inset. However, we can still define *quasi*-pillow patches and *quasi*-pillow blocks; these have more than just two bounding cells, but have only two remaining opposite sides under the T-mesh labeling $\mathcal{L}$. The tesselation of the remaining two sides is not necessarily equivalent, but the coincidence relations derived from the quantization prescribe how the two sides must be merged.

In section 9.3.1 we first consider the abstract connectivity aspects of simple atomic collapses. Afterwards, in section 9.3.2, the accompanying treatment of the complex's embedding in a background mesh is addressed. Finally, in section 9.4 we describe an approach that decomposes quasi-pillow collapses into multiple atomic collapses—after appropriate local refinement.

9.3.1 Connectivity Collapse Operators

We exploit a symmetry among the above three simple operators to simplify exposition: In each case, an n-cell c_0 incident to one or two $(n-1)$-cells c_1 and c_2 is to be collapsed. fig. 9.4 illustrates this situation (for the case of *two* incident cells, c_1 and c_2) using the relevant excerpt from the incidence graph defined by the meta-mesh's incidence relation (cf. theorem 2.3).

Assume, the collapse of n-cell $c_0 \in \mathcal{T}$ bounded by two $(n-1)$-cells $\partial c_0 = \{c_1, c_2\}$ is to be performed. This kind of collapse is a directed operation, meaning c_1 is collapsed into c_2 or vice-versa, resulting in either c_1 or c_2 vanishing. We assume c_1 is supposed to vanish in the following. If $\partial c_0 = \{c'\}$, we denote $c_1 = c_2 = c'$ to unify exposition. Because n-cells are topological n-balls, we know that $\partial c_1 = \partial c_2$.

The updates of incidence relations ∂, performed to execute the collapse of c_0 are summarized in algorithm 9.1. Here, multiset logic is employed to handle also cases of self-adjacency (as introduced in section 6.2.1). Line 6 are intentionally separate to cover the case where $c_1 = c_2 = c'$. Because ∂^{-1} is inferred from ∂ it needs no explicit update.

As bounds are removed from cells $w \in \partial^{-1} c_0$ and $x \in \partial^{-1} c'$, these cells could hypothetically become isolated, i.e., $\partial x = \emptyset$ or $\partial w = \emptyset$. However, for this to occur, such cells must have had only a single bound to begin with. By always collapsing singly-bounded cells prior to their bound, a sane connectivity is remained throughout. Specifically, in fig. 9.3(c) and (e) the connectivity collapse is performed top-down, collapsing block before patch before arc.

Let us remark that a collapse of a loop arc that is part of the boundary of a block but does not bound a patch of that boundary, while possible in terms of connectivity, would result in a 'pinched' block, that would no longer be an open n-ball. Also non-pure complexes could result, with an arc not incident to any patch or a patch not incident to any block. Our application scenario does not require any such collapses; we ensure the complex remains pure and composed of open n-balls.

Algorithm 9.1:
CollapseConnectivity(c_0)

Input: Cell to collapse c_0
Output: Updated $\mathcal{T}$, ∂

$(c_1, c_2) \leftarrow$ chooseDirection(∂c_0)

// Remove c_0 //
foreach $w \in \partial^{-1} c_0$ **do**
  $\partial w \leftarrow \partial w \setminus \{c_0\}$
$\partial c_0 \leftarrow \emptyset$
delete c_0 from $\mathcal{T}$

// Replace c_1 by c_2 //
foreach $x \in \partial^{-1} c_1$ **do**
1 $\partial x \leftarrow \partial x \uplus \{c_2\}$
foreach $x \in \partial^{-1} c_1$ **do**
2 $\partial x \leftarrow \partial x \setminus \{c_1\}$
$\partial c_1 \leftarrow \emptyset$
delete c_1 from $\mathcal{T}$

9.3.2 Embedding Collapse Operators

When collapsing cells of complexes that are embedded, the embedding needs to be updated in addition to the connectivity updates. Otherwise, when just restricting the map $\mathcal{I}$ to the remaining cells, it would no longer satisfy the embedding conditions stated in section 6.2.4. In particular, structure preservation and surjectivity are at stake. In this section we describe, for each atomic

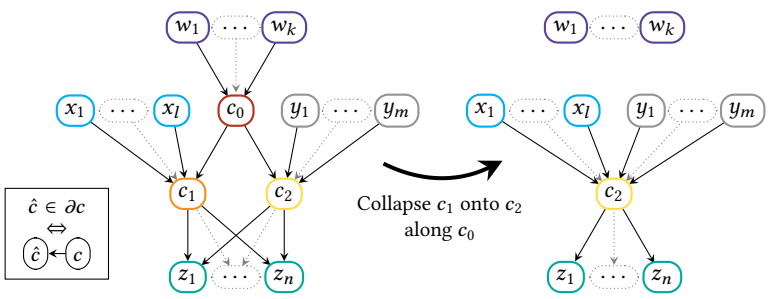


Figure 9.4: Illustration of a dimension-generic collapse in terms of the incidence graph, showing the state before (left) and after (right). Cells incident to neither c_0, c_1 nor c_2 are unaffected and thus omitted. Layers above c_0 or below c_1, c_2 are potentially empty.

collapse operator, the discrete embedding updates to be performed (in conjunction with the connectivity changes discussed above), so as to maintain a valid embedding. For simplicity, we do not explicitly discern between a cell and its embedding image in the following, so e.g., a patch p and the set of facets $\mathcal{I}_p = \{f_0, f_1, \dots\}$ it is embedded into are both referred to as patch in the following.

Erase-and-Recompute vs. Incremental A potential approach would be to discard the embedding of all cells (nodes, arcs, patches, and blocks) directly adjacent to the removed cell, and then to recompute suitable embedding images for these. Constructing in particular the required discrete facet surfaces, to serve as embedding images for patches, subject to manifoldness, genus, and homotopy constraints, guaranteeing correctness even in the presence of self-adjacent cells, is a challenging endeavour in the three-dimensional setting. We therefore develop a strategy to instead *maintain* embedding validity. It is based on incrementally transforming the former embedding using a set of suitable operators.

Operator choice Note that the set of possible structural configurations around a to-be-collapsed cell is very diverse. The number of patches incident on an arc is varying, the number and adjacency pattern of arcs incident on a node is varying, subsets of incident cells can lie in the boundary, cells can be self-adjacent in multiple ways, cells are embedded in varying numbers and configurations of background mesh elements, and so forth. To deal with this complexity, we break the task down to very simple local operators. First of all, we proceed *incrementally*; for instance, in the context of an arc collapse, we focus on the problem of moving a node along a single edge rather than along an entire arc (embedded in a path of edges) at once. Furthermore, we build the operators *recursively*, such that updating a cell c incurs updates of other cells bounded by it (and so on) until the entire star c^* is consistent again. Together, this allows us to restrict ourselves to just three local base operators, which we can then compile into the higher-level operators needed to accompany the above topological collapses.

9.3.2.1 Mental Picture

Topologically, n-cells are open n-balls: arcs are line segments, patches are disks, and blocks are balls. In the collapse scenario, barring self-adjacency, these are bounded by two $(n-1)$-bounds, forming two halves of an n-sphere. In this case one of the half-n-spheres must be collapsed onto the other half, conceptually by shifting it over the interior of the bounded n-ball and squishing the n-ball to zero extent in the process. Under self-adjacency, for $n > 1$ it is possible the whole n-sphere is covered by only a single, self-adjacent bound. In this case, the mental picture corresponds to a contraction of this single n-spherical bound onto its own bounds, again deflating the n-ball in the process.

Assume more concretely, a c_0 with two distinct bounds c_1 and c_2 is to be collapsed and c_1 to be replaced by c_1. Then, the embedding update's main goal is to shrink c_0 to zero extent, by shifting c_1 across it and towards c_2. In technical terms, c_1's embedding should be homotoped into that of c_2 through the interior of c_0—in discrete steps crossing a background mesh cell in each step. While doing so, the cells bounded by c_2 must expand and shift accordingly, to take up this space and reestablish a structure-preserving state. Through transitivity, this means that exactly the cells in the star c_1^* of c_1 require updates (theorem 2.5). In this way topological consistency between connectivity and (embedding-induced) geometry is maintained. Note that updating the embedding for all of c_1^* has further-reaching effect than the connectivity updates (algorithm 9.1), for which only updates to $\partial^{-1}c_1$ were necessary.

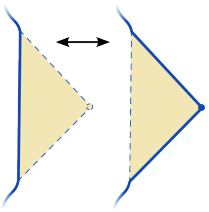
In the case that c_0 is only bounded by a single cell c_1 one instead contracts c_1's image towards its bound(s) ∂c_1, again pulling any cells from c_1^* with it to maintain correspondence between connectivity and geometry.

Complicating factors The collapse situation generally can be slightly more constrained than detailed above, especially in the presence of meta-mesh cells representing critical entities. These must be preserved, hence their embedding may only be collapsed *within* their containing critical entity. Furthermore, the degrees of freedom offered by the background tet mesh may not be sufficient to shift a cell's embedding while keeping it disjoint from others, requiring background mesh refinement. Lastly, the geometry of both background cells and meta-cells might become severely distorted or near-degenerate in the collapsing process, posing numerical challenges for applications of the collapsed complex. For a simpler exposition we first assume absence of such complicating conditions here and cover these afterwards in section 9.5.

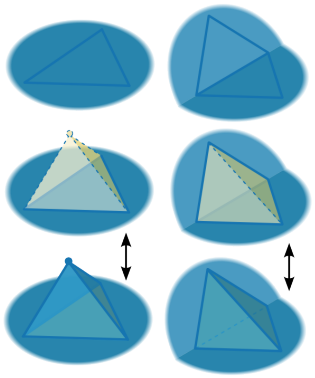
9.3.2.2 Base Operators

NODESHIFT This operator simply moves a node n across one of its incident edges. If the traversed edge belongs to some arc a, then it is removed from a, meaning a is shortened by one edge on one of its ends. Note that this base operator alone does not maintain embedding validity if further arcs are incident. Rather, it is used in the following sections to compose higher-level operators in a recursive manner.

ARCSHIFT This operator shifts a segment of an arc a by routing it around the other side of a single triangle f incident to the arc's path. Effectively, this operation toggles the edges of triangle f into or out of a: those that were part of a before are removed from it, while those that were not are inserted. In case f is part of a patch p, then it is removed from p. Again, this base operator alone does not yet maintain embedding validity in general.



PATCHSHIFT Analogously, a single patch p can be altered by routing the patch around the other side of a single tetrahedron c incident to the patch surface. Again, facets previously part of p are removed from it, while others are inserted into it. Additionally, because a (non-boundary) patch always separates two blocks b_1 and b_2 we can easily update the blocks' embedding, by logically moving the traversed cell c from one block into the other. The PATCHSHIFT operator applied on a non-boundary patch maintains a valid embedding, as it includes compatible updates also of the embedding of the incident higher-dimensional cells (two blocks). It can therefore be applied in a standalone manner.



ARCSHIFT and NODESHIFT, by contrast, cannot. They disconnect an arc from its incident patches or a node from its incident arcs in terms of their embedding, respectively. Hence, we define an ARCPATCHSHIFT, essentially an ARCSHIFT followed by PATCHSHIFTS that pull the incident patches with it, so as to maintain a consistent embedding. Similarly, a NODEARCPATCHSHIFT is defined, essentially a NODESHIFT followed by ARCPATCHSHIFTS for the incident arcs. Note that this effectively forms a recursion along the (transitive) bottom-up incidences of the node: First the node is shifted, which triggers shifts for the incident arcs, each of which triggers shifts for the incident patches (which include a built-in shift for their incident blocks).

9.3.2.3 ArcPatchShift Operator

Let us first consider only a single patch p incident to the arc a to be shifted, as displayed in fig. 9.5. Initially a is lifted across the beige triangle by an ArcShift and as a consequence is disconnected from p. To recover valid connectivity, the traversed triangle is inserted into p. As patch overlaps are to be avoided for embedding injectivity, this alone is not sufficient: there is only one traversed triangle but possibly multiple incident patches to be reconnected. To free up space for subsequent patches, p is lifted off the traversed triangle (and the former arc edge, for that matter) by means of PatchShift operators.

In the simple case displayed in fig. 9.5, no more than a single shift across one tetrahedron is needed. In the more general case however, lift-off may require multiple shifts in sequence, traversing a whole sector of tetrahedra in the process. The cross-section of such a more general case (also involving multiple incident patches) is shown in fig. 9.6. In this example it also becomes clear that the order in which patches are processed is relevant. It is necessary to "peel off" the upper layer of patches before gaining access to the next lower layer. For instance, the patch

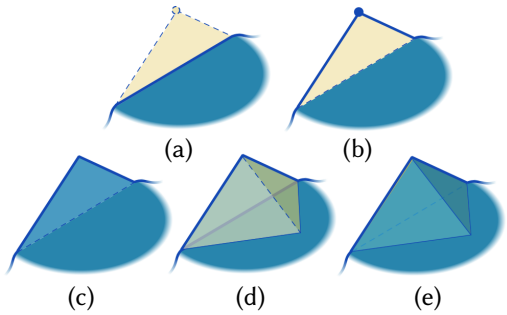


Figure 9.5: Illustration of sub-steps of (part of) an ArcPatchShift. After an arc (dark blue) is shifted across a single facet (beige) via an ArcShift, one patch (blue) incident to the arc is first reconnected to the shifted arc by appending the traversed triangle into the patch. Then it is lifted off by (one or more, here one) PatchShifts to resolve overlaps with other patches incident to the arc. Figure 9.6 illustrates how multiple incident patches are handled sequentially.

Algorithm 9.2:
ArcPatchShift(a, f)

Input: Background mesh $\mathcal{M} = \{T, F, E, V\}$, meta-mesh $\mathcal{T} = \{B, P, A, N\}$, embedding $\mathcal{I}_{\mathcal{T}} \hat{=} \mathcal{I}_{\mathcal{M}}^{-1}$, arc $a \in A$, incident face $f \in F$
Output: Consistent $\mathcal{I}_{\mathcal{T}}$ with a shifted across f

```
// E_a: Edges removed from I(a) //
E_a ← ∂f ∩ I(a)
ArcShift(a, f)
// Through block layer on f… //
foreach block b ∈ I⁻¹(∂⁻¹f) do
    // …reach patches on E_a //
    while ∃p ∈ (∂b ∩ I⁻¹(∂⁻¹E_a)) do
        // Last successor gets f //
        if I⁻¹(∂⁻¹E_a) = {p} then
1           I⁻¹(f) ← p
            return
        // Shift rest off E_a via b //
        while ∃f' ∈ (I(p) ∩ ∂⁻¹E_a) do
            t ← t ∈ (I(b) ∩ ∂⁻¹f')
            PatchShift(p, t)
2           I⁻¹(f) ← 0
```

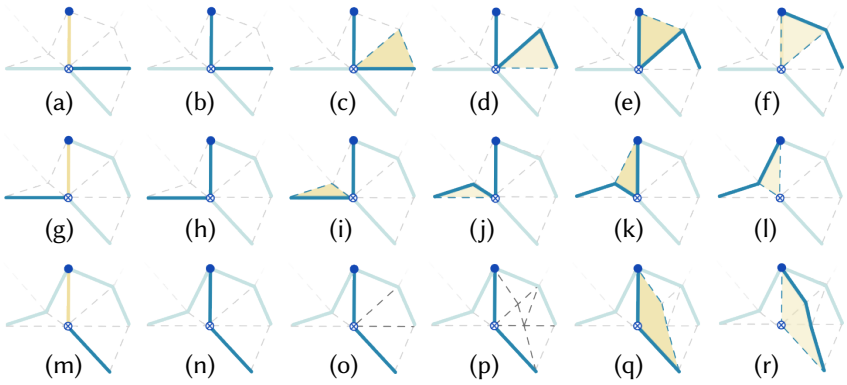


Figure 9.6: Cross-section illustration of a ArcPatchShift. The first row corresponds to fig. 9.5, except that here two PatchShifts are required to lift off the first incident patch. The further rows show the handling of the second and third incident patch. Note that in the end the surrounding wheel of patches of the shifted arc is properly connected to it again.

pointing down-right in fig. 9.6 can only be shifted after one of the two patches pointing left and right. Also, a situation in which the background mesh resolution is not sufficient is evident in fig. 9.6o, with fig. 9.6p indicating the local background mesh refinement necessary to allow the last patch to be shifted. This is discussed in detail in section 9.5.3. Note that the last patch shift in fig. 9.6o-r may be skipped altogether, as fig. 9.6n is already a valid ending configuration; this is discussed further in section 9.5.6.

Algorithm 9.2 details the ARCPATCHSHIFT in formal notation. Note that there is a slight difference between the pseudocode and the depictions shown before: The traversed facet is not inserted into the follow-up patch beforehand (only to be toggled out of it with the last ARCSHIFT), but instead is toggled in with the final ARCSHIFT and erased again afterwards (line 7). Both variants are equivalent, but the latter allows for a more concise notation. For the same reason line 7 alter the inverse embedding only, implying that the forward embedding is updated accordingly.

9.3.2.4 NODEARCPATCHSHIFT Operator

Consider the situation in fig. 9.7a: A node, within one block, is incident to three incident arcs and three incident patches. It is to be shifted by one edge along one of the incident arcs. Firstly, the node itself is shifted by a NODESHIFT, which disconnects it from two of the surrounding arcs. As a consequence, the arcs as well have to be shifted, one after the other, and proceeding layer by layer. Within the single layer shown in fig. 9.7, one such arc is chosen and arc-node connectivity recovered by appending the traversed edge into this arc (fig. 9.7c-d). Using a sequence of ARCPATCHSHIFTS along a triangle fan (fig. 9.7e) the arc is lifted off the former node vertex (and thus off the traversed edge as well) and incident patches are pulled with it. This procedure is

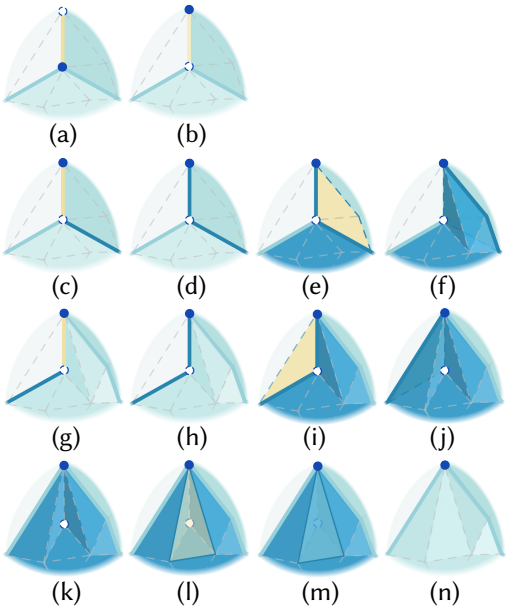


Figure 9.7: Illustration of sub-steps of (part of) a NODEARCPATCHSHIFT. Arc by arc, ARCPATCHSHIFT operators are applied to reconnect these arcs with a shifted incident node. In the end (last row) a PATCHSHIFT is applied to lift the blue patch off of the former node vertex (white).

Algorithm 9.3:
NODEARCPATCHSHIFT(n,e)

Input: Background mesh $\mathcal{M} = \{T, F, E, V\}$, meta-mesh $\mathcal{T} = \{B, P, A, N\}$, embedding $\mathcal{I}_\mathcal{T} \hat{=} \mathcal{I}_\mathcal{M}^{-1}$, node $n \in N$, edge $e \in E$
Output: consistent $\mathcal{I}_\mathcal{T}$ with n shifted across e

$v = \mathcal{I}(n)$
NODESHIFT(n, e)

```
// Through block layer on e… //
foreach block b ∈ I⁻¹(∂⁻²e) do
    // …via patches on e… //
    foreach patch p ∈ (∂b ∩ I⁻¹(∂⁻¹e)) do
        // …reach arcs on v… //
        while ∃a ∈ (∂p ∩ I⁻¹(∂⁻¹n)) do
            // …shift them off v via p //
            while ∃e′ ∈ (I(a) ∩ ∂⁻¹n) do
                f ← f ∈ (I(p) ∩ ∂⁻¹e′)
                ARCPATCHSHIFT(a, f)
1               I⁻¹(e) ← 0
    // …reach patches on n… //
    while ∃p ∈ (∂²b ∩ I⁻¹(∂⁻²n)) do
        // …and shift them off v via b //
        while ∃f′ ∈ (I(p) ∩ ∂⁻¹n) do
            t ← t ∈ (I(b) ∩ ∂⁻¹f′)
            PATCHSHIFT(p, t)
```

repeated for any arcs remaining in the current layer (fig. 9.7g-j). Afterwards, if a patch of the current layer is still incident to the former node vertex, it is lifted off by a sequence of PATCHSHIFTS (fig. 9.7k-n). While fig. 9.7 shows only a single layer, the processing order of further layers is analogous to the procedure in ARCPATCHSHIFT, repeatedly lifting the uppermost layer (reachable from the collapse edge) off the former node vertex and thereby making the next layer accessible for lift-off until none is left.

Algorithm 9.3 details the NODEARCPATCHSHIFT in formal notation. Note that, here again, the pseudocode differs from the illustrations in that it does not append the traversed edge into the follow-up arcs beforehand (which would result in the arc getting toggled *out* with the last ARCPATCHSHIFT) but instead removes it after it has been toggled *in* with the final ARCPATCHSHIFT (line 8). Note that both algorithms 9.2 and 9.3 contain unordered loops over incident elements. These can indeed be processed in arbitrary order, this has only geometric effects on the resulting embedding. However, it is important that **foreach** loops re-check for new iterable elements after each iteration, as new ones may become available through instructions in the loop body.

9.3.2.5 Complete Operators

Using merely the above two operators, NODEARCPATCHSHIFT and ARCPATCHSHIFT, we can now define the embedding updates to accompany the different types of collapses.

Arc Embedding Collapse Updating the embedding for the collapse of an arc with two incident nodes now is easy: one end node's embedding is removed and then a sequence of NODEARCPATCHSHIFTS is executed, for each edge along the arc, starting from its other end. Figure 9.8 shows such a collapse of an arc composed of three edges. A loop-arc can only be collapsed if it bounds a patch, as noted in section 9.3.1, hence we handle it as a patch collapse detailed below.

Patch Embedding Collapse The collapse of a pillow patch p, i.e., a patch incident to exactly two arcs, is slightly more involved. Similar to the arc collapse, first one arc a_2 of the patch is chosen, its embedding erased, and then the remaining arc a_1 is incrementally shifted across p by a sequence of ARCPATCHSHIFTS: Any facet of $\mathcal{I}_p$ incident to at least one edge $\mathcal{I}_{a_1}$ is chosen, a is shifted across that facet and the process is repeated until $\mathcal{I}_p = \varnothing$ and $\mathcal{I}_{a_1}$ coincides with the former $\mathcal{I}_{a_2}$. This process is demonstrated in fig. 9.9 for a case where there is only one patch dragged with the shifting arc and with shifts applied batch-wise for brevity. For the special case of a patch with only a single bounding loop-arc a_2 the procedure is similar: a_1 is incrementally shifted across p by a sequence of ARCPATCHSHIFTS, until both vanish, i.e., $\mathcal{I}_p = \mathcal{I}_{a_1} = \varnothing$. The case of a self-adjacent patch collapsing, i.e., one that covers the complete boundary of a block, is covered by the block collapse detailed below.

It is sensible in the above process to postpone shift operations that would temporarily introduce a cycle into $\mathcal{I}_a$, as special case handling in ARCPATCHSHIFT and unnecessary refinement can be

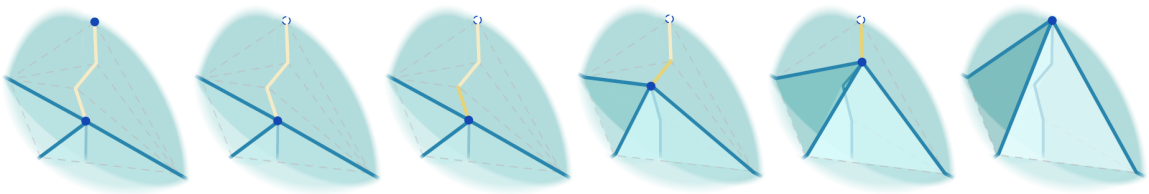


Figure 9.8: When topologically collapsing an arc (yellow) the complex's embedding can be updated through (after deleting one incident node from the embedding map) a sequence of NODEARCPATCHSHIFTS, edge by edge.

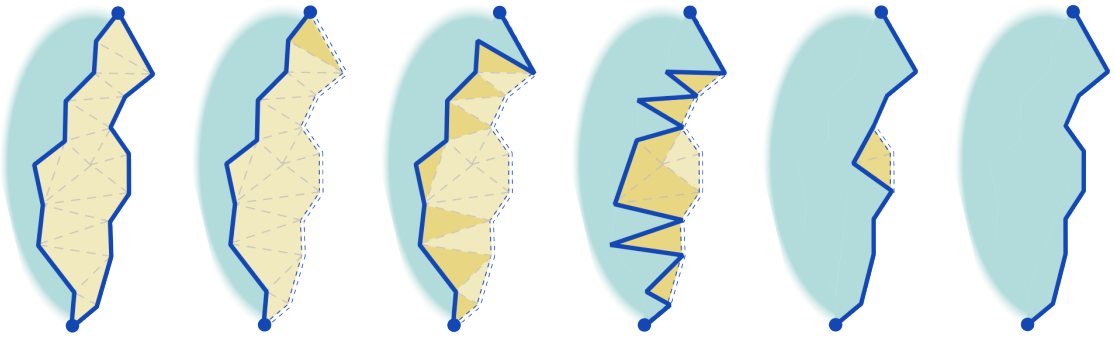


Figure 9.9: The embedding update for a pillow patch collapse is performed (after deleting one incident arc from the embedding map) by a sequence of ARCPATCHSHIFTS, across all triangles of the patch (yellow).

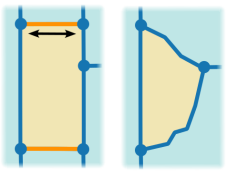
avoided that way at no additional cost. Such a topology-preserving ordering of shifts is possible because disk-topology triangle meshes (and thus the patch image) are extendably shellable [Bruggesser and Mani 1971].

Block Embedding Collapse While the collapse of a block b with one or two incident patches (that have common boundaries) could be carried out in an incremental way in analogy to the above patch collapse (using PATCHSHIFTS), this effort is not actually necessary. Because an incident (non-boundary) patch is incident to only one further block, there is no need for the non-trivial procedure of dragging multiple successors behind it—as was necessary for the above arc and patch collapse operators. Instead, we only need to erase an incident patch p from the embedding map, and move all cells from $\mathcal{I}_b$ over to $\mathcal{I}_{b'}$, where b' is the other block that was incident to p.

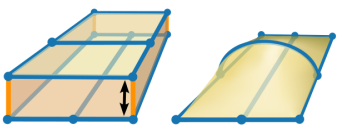
9.4 Collapsing Quantized T-Meshes

The previous sections define the operators necessary to perform collapses in general embedded cell complexes. We now extend this set of operators by additional collapse operators for pillow patches with more than two incident arcs and for pillow blocks with more than two incident patches. While in general this comes with ambiguities regarding the resulting connectivity, we consider here patches and blocks in T-meshes, i.e., non-conforming cubical complexes as defined in section 9.2. As noted in section 9.3, there is an analogue of the aforementioned pillow cells on general to T-meshes, which we call *quasi-pillow* patches and blocks. Concretely, these have exactly two remaining opposite sides under the T-mesh labeling, each potentially consisting of multiple cells.

Any arbitrarily complex patch that should be collapsed becomes a quasi-pillow patch by collapsing all arcs on two opposite sides that are zero-sides. We devise a method to collapse such quasi-pillow patches in the following. An unambiguous way to merge the tesselations of the remaining two sides is deduced from the underlying quantization.

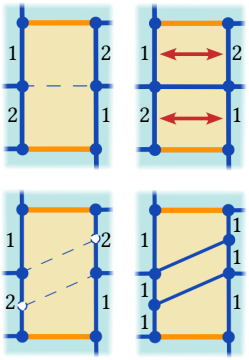


Similarly, any blocks that should be collapsed become quasi-pillow blocks by collapsing all arcs and patches on the four sides aligned with a zero-extent direction of the block. We also devise a method to collapse such quasi-pillow blocks in the following, with an equivalent merging strategy for the remaining two sides.

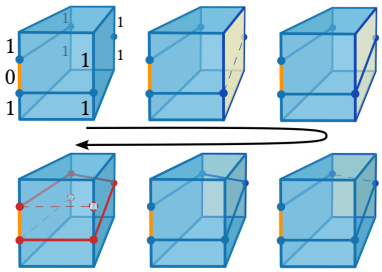


9.4.1 Reduction to Conformity?

It is tempting to try to reduce the non-conforming situation to a conforming one. Then the known operators could be applied. This, however, is not easily possible in general, especially when the complex needs to remain consistent with the assigned quantization, as the following considerations show.



An intuitive idea might be to, if the T-mesh was created such that it is aligned with a parametrization of the background mesh, simply split patches and blocks by extending all T-junctions by tracing isolines and isosurfaces. This generates additional parametrization-aligned arcs and patches, splitting the blocks into a conforming state. Alas, splitting patches or blocks by these may contradict and invalidate the existing assigned quantization, creating patches whose opposite side lengths do not match (and which therefore are not usable in the hexahedral mesh generation context). Indeed, splits rather must be aligned with respect to the *quantization*.



Trying to split all zero-patches and zero-blocks using a quantization-aligned splitting strategy also proves to be difficult. Consider a block with a single zero-arc. Trying to split all its patches in a quantization-aligned way is an ambiguous task—due to the zero-arc separating two layers of arcs that are actually at the same "height" in terms of the quantization. This easily leads to infinite splitting spirals, as illustrated in the inset.

9.4.2 Bisection Operators

For these reasons, we introduce additional operators to split, or more precisely bisect, arcs, patches and blocks. This then allows us to perform collapses of quasi-pillow patches by means of a suitably chosen sequence of bisections, arc collapses, and pillow patch collapses; quasi-pillow blocks can be collapsed by means of a sequence of bisections, pillow-patch collapses, and pillow-block collapses. Effectively, the collapse of the complex quasi-pillow cells is reduced to a sequence of simple operations.

Arc Bisection As a prerequisite for the bisection of *quasi-pillow* patches and blocks we require first of all an arc bisection operation. More specifically, it is required that an arc can be split at a certain (quantized) distance s along its length. A new node is inserted (and embedded) at that distance along the arc, and the arc is split into two sub-arcs at the new node. One of these then has the desired quantized length s. The sub-arcs each receive one segment of the original arc's embedding, with segments separated by the vertex into which the new node is embedded. Algorithm 9.4 details the execution of the arc bisection operator.

Patch Bisection Recall that *quasi-pillow* patch is a patch that only has two opposite sides but is bounded by more than two arcs. We handle quasi-pillow patches by bisecting them at a T-junction, by means of introducing a zero-arc. This zero-arc can then be collapsed using the known operator right away, yielding two simpler (quasi-)pillow patches. Recursive application of this bisection operator eventually yields pillow patches (without any T-junctions), that can then be collapsed. Figure 9.10 illustrates this.

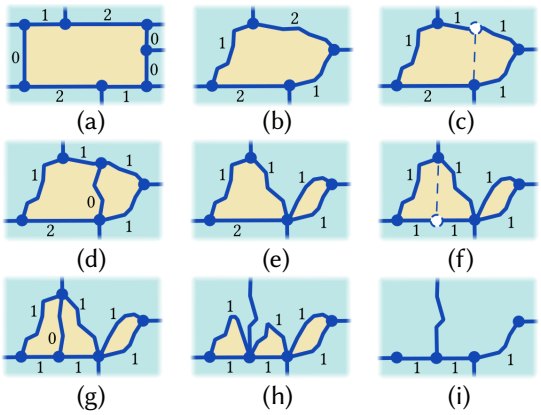


Figure 9.10: From an initial configuration (a), a quasi-pillow patch results after performing zero-arc collapses (b). This allows it to be collapsed by means of simple patch collapses (h-i), after recursively bisecting it (d,g) by extending a T-junction (c,f).

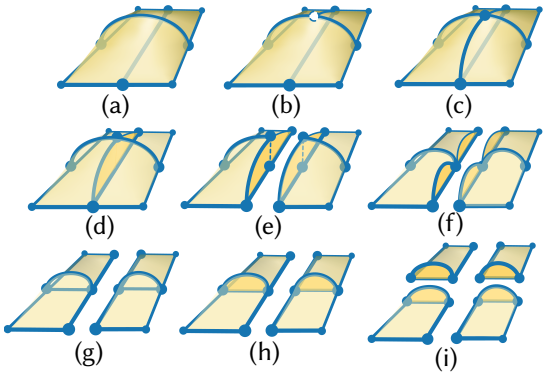


Figure 9.11: The boundary of a quasi-pillow block (a) is refined by extending a T-arc (bottom side) around the block, patch by patch (b) until it forms a cycle (c). Then a patch is inserted, bounded by this cycle, bisecting the block (e). Quasi-pillow-patch collapses (e-f) and then pillow-patch collapses (f-g) become possible. Recursive block bisection (g-i) in the end leaves simple pillow-blocks.

Algorithm 9.4: BISECTARCAT(a,n,s)

Input: Aligned T-mesh $\mathcal{T} = \{B, P, A, N\}$, embedding $\mathcal{I}_\mathcal{T}$, quantization q, arc a, node n, segment length $q_s \in \mathbb{Z}$
Output: a bisected, sub-arc $a_1 \in \partial^{-1} n$ with $q_{a_1} = q_s$

$r \leftarrow q_s / q_a$
$v \leftarrow$ vertex at relative length $\sim r$ along $\mathcal{I}(a)$ from n
$n_{\text{new}} \leftarrow$ new node
$\mathcal{I}(n_{\text{new}}) \leftarrow v$
$\{n'\} \leftarrow \partial a \setminus \{n\}$
$(a_1, a_2) \leftarrow$ bisect a s.t. $\begin{cases} \partial a_1 \leftarrow \{n, n_{\text{new}}\}, \\ \partial a_2 \leftarrow \{n', n_{\text{new}}\} \end{cases}$
$(\mathcal{I}(a_1), \mathcal{I}(a_2)) \leftarrow$ partition $\mathcal{I}(a)$ by v
$(q_{a_1}, q_{a_2}) \leftarrow (q_s, q_a - q_s)$

Algorithm 9.5: BISECTPATCH(p)

Input: Aligned T-mesh $\mathcal{T} = \{B, P, A, N\}$, embedding $\mathcal{I}_\mathcal{T}$, quantization q, quasi-pillow patch p
Output: p bisected

$\{A_\uparrow, A_\downarrow\} \leftarrow$ opposite sides of p with $|A_\uparrow| + |A_\downarrow| > 2$
$\{a_\uparrow, a_\downarrow\} \leftarrow$ first arcs of $A_\uparrow, A_\downarrow$ on one side of p
if $q_{a_\uparrow} < q_{a_\downarrow}$ **then** swap$(a_\uparrow, a_\downarrow)$
if $q_{a_\uparrow} > q_{a_\downarrow}$ **then**
  $n_0 \leftarrow$ patch corner $n \in \partial a_\uparrow$
  $a_\uparrow \leftarrow$BISECTARCAT$(a_\uparrow, n_0, q_{a_\downarrow})$
$\{n_\uparrow, n_\downarrow\} \leftarrow$ non-corner nodes $n \in \partial\{a_\uparrow, a_\downarrow\}$
$\{A_1, A_2\} \leftarrow$ partition ∂p by $n_\uparrow, n_\downarrow$
$a_{\text{new}} \leftarrow$ new arc with $\partial a_{\text{new}} = \{n_\uparrow, n_\downarrow\}$
$q_{a_{\text{new}}} \leftarrow \text{dist}_q(n_\uparrow, n_\downarrow)$
$\mathcal{I}(a_{\text{new}}) \leftarrow \bigcup_{a' \in A_1} \mathcal{I}(a')$
while $\mathcal{I}(a_{\text{new}})$ overlaps with any $\mathcal{I}(a')$ **do**
  foreach facet $f \in \mathcal{I}(p)$ incident to the overlap **do**
    ARCSHIFT(a_{new}, f)
$(p_1, p_2) \leftarrow$ bisect p s.t. $\begin{cases} \partial p_1 \leftarrow A_1 \cup \{a_{\text{new}}\}, \\ \partial p_2 \leftarrow A_2 \cup \{a_{\text{new}}\} \end{cases}$
$(\mathcal{I}(p_1), \mathcal{I}(p_2)) \leftarrow$ partition $\mathcal{I}(a)$ by $\mathcal{I}(a_{\text{new}})$

The introduction of the arc that bisects the patch requires some care. The point that the arc is connected to on the side opposing the T-junction (white in fig. 9.10) needs to be chosen within the correct arc (or on the correct node), based on the assigned quantization, maintaining equal values on the opposite sides (fig. 9.10c,f). As discussed above, this arc or node may not be unique if there are zero-arcs on the opposite side. We say a patch bisection operation is *executable* if no such zero-arcs cause an ambiguity.

Once two nodes that are opposite under q have been determined (or created by arc bisection), an arc connecting the two is inserted. To determine a suitable embedding for this arc in a topologically safe manner, we initialize it with one half of the patch boundary (ending at the two opposite nodes), and then pull this arc off of the arc boundary into the patches's interior using the ARCSHIFT operator. Algorithm 9.5 details the execution of the patch bisection operator assuming executability. It describes a more general application of patch bisections, applicable also to non-collapsing patches, as this is needed to reconcile opposite side tesselations for the block bisection described next.

9.4.2.1 Block Bisection

A quasi-pillow block is a block that has only two opposite sides but more than two patches, i.e., sides are partitioned by some T-junction arcs (fig. 9.11a) due to patches being incident from the block's outside. Similar to the patch bisection case, our goal is to split the block into two by extending a T-junction—which here, however, is an arc in a block side rather than a node. Extending it requires the insertion of an additional patch, splitting the block while containing the T-arc in its boundary.

To this end, we first determine the boundary (a cycle of arcs) of the patch to be inserted. If the arc is already part of a cycle of arcs (in a common plane in terms of the quantization) on the block's boundary, we are done. Otherwise, it is part of a partial cycle (potentially just the single arc) that ends in a T-node at a patch side on the block boundary (fig. 9.11a). We extend this T-junction across the subsequent patch (via the patch bisection operation discussed above). This is repeated until the arc cycle is closed (fig. 9.11b-c).

Once this cycle of arcs is determined, we can insert a new patch, bisecting the block, incident to this boundary cycle (fig. 9.11d). To determine a suitable embedding for this patch in a topologically safe manner, we initialize it with one half of the block boundary, bounded by the arc cycle, and then pull this patch off of the block boundary into the block's interior using the PATCHSHIFT operator.

Again, we call a block bisection *executable* if it involves no ambiguities due to zero-elements. Algorithm 9.6 of the block bisection operator, assuming executability. For simplicity of exposition, the latter algorithm initially extends *all* T-nodes on the block boundary (rather than focusing on creating the cycle for one specific T-arc only). In subsequent recursive bisections of sub-blocks then no further such extensions are necessary.

Algorithm 9.6: BISECTBLOCK(b)

Input: Aligned T-mesh $\mathcal{T} = \{B, P, A, N\}$, quantization q, quasi-pillow block b
Output: b bisected

while $\exists p \in \partial b : |\partial p| > 4$ **do**
  BISECTPATCH(p)
$A^\circ \leftarrow$ any cycle $A \subset \partial^2 b$ with constant q-alignment
$\{P_1, P_2\} \leftarrow$ partition ∂b by A°
$p_{\text{new}} \leftarrow$ new patch with $\partial p_{\text{new}} = A^\circ$
$\mathcal{I}(p_{\text{new}}) \leftarrow \bigcup_{p' \in P_1} \mathcal{I}(p')$
while $\mathcal{I}(p_{\text{new}})$ overlaps with any $\mathcal{I}(p')$ **do**
  foreach tet $t \in \mathcal{I}(b)$ incident to the overlap **do**
    PATCHSHIFT(p_{new}, c)
$(b_1, b_2) \leftarrow$ bisect b s.t. $\begin{cases} \partial b_1 \leftarrow P_1 \cup \{p_{\text{new}}\}, \\ \partial b_2 \leftarrow P_2 \cup \{p_{\text{new}}\} \end{cases}$
$(\mathcal{I}(b_1), \mathcal{I}(b_2)) \leftarrow$ partition $\mathcal{I}(b)$ by $\mathcal{I}(p_{\text{new}})$

9.4.3 Global Collapse Strategy

Processing the zero-cells of a complex that are to be collapsed in order of increasing dimension (collapse arcs before patches before blocks) would trivially ensure executability of the required bisection operations. However, this strict prioritization of lower-dimensional cells can lead to special configurations. One example is the case of a block b, for which all arcs $a\partial^2 b$ are zero-arcs; collapsing all such zero arcs first would leave a 'balloon'-block, that has only a single node on its boundary (incompatible with our incidence relations theorem 2.2). This would not only require the definition and implementation of additional embedding-maintaining collapse operators for these special configurations but also a data structure that can handle these intermediate configurations. While certainly possible, we can instead avoid these by rather giving priority to collapses of higher-dimensional cells *if* they are executable, i.e., whenever there is no ambiguity. In the above example, prior to the collapse of the last edge for instance, the block is one of type fig. 9.3e, which is covered by sections 9.3.1 and 9.3.2.5, as well as by our data structure.

Further examples of cases that can be avoided if we proceed in this way are:

- A patch with only a single node on its boundary, as it would bound a block like in the above example, which is precluded.

- A loop-arc of zero quantized length; this could result only from the collapse of one arc in a cycle of two zero-arcs. This two-arc loop either bounds a zero-patch, in which case it is handled as a pillow-patch collapse, or does not already bound one, in which case one will be inserted via the bisection of a quasi-pillow block the two-arc loop encloses. Note that such an enclosed block must exist, because a valid quantization creates no zero-length incontractible cycles.

Concretely, the specific order in which we apply the previously introduced structure-preserving collapse and bisection operators to all zero-extent cells is:

- Repeat until no zero-extent cells remain, giving priority to operators as follows:
  1. pillow-block collapse
  2. pillow-patch collapse
  3. executable zero-block bisection
  4. executable zero-patch bisection
  5. zero-arc collapse

Note that if the zero-labels are determined by a *valid* quantization, all zero-elements will eventually be collapsed.

Ordering Within each of the above priority classes, the order of operations can be chosen arbitrarily. The connectivity of the ultimately resulting complex is already uniquely determined by the given quantization. What, however, is affected—by the choice of *order* as well as the choice of collapse *direction*—is its concrete geometric embedding. For our purpose, this is of limited relevance: It will affect the distortion of the initial IGM built out of the block-wise maps, but this is globally optimized afterwards anyway, i.e., it only serves as initialization. Nevertheless, distorted blocks may increase the hardness of the mapping and optimization problem. We therefore evaluated multiple ordering strategies:

1. random order, random direction
2. smallest first, least-effort direction
3. smallest first, coordinated direction

'Least-effort direction' means that of the two possible collapse directions of an element, the one that requires the smaller number of incident elements to be re-embedded is chosen. 'Coordinated direction' means that the direction for arc collapses is not chosen individually per arc but consistently for sequences of arcs that are adjacent via patches; this avoids zig-zag behavior when sheets of multiple adjacent blocks or patches are to be collapsed. According to our experiments, strategy (2) is beneficial over strategy (1), and strategy (3) leads to geometrically even more favourable embeddings, so we make use of this in our application scenario.

9.5 Intricacies

In the previous sections we set aside the discussion of some practically relevant aspects for a more concise presentation of the (admittedly still quite extensive) core collapsing method. We address these deliberately neglected aspects in the following, starting with those applicable to general embedded cell complexes and touching on those relevant specifically for T-mesh

collapsing afterwards. To address implementation aspects that cannot be covered in full detail here, we released a reference implementation of all described operators, their strategic application to collapse T-meshes and some (limited) map optimization machinery to compute injective integer-grid maps on the resulting collapsed T-meshes, in form of the open source library C4HEX[1].

9.5.1 Self-Adjacent Cells

In the description and notation of algorithms we have assumed unambiguity in the connectivity of patch and block boundaries; for instance each patch corner was assumed to have exactly two perpendicular outgoing arcs, three in case of a block corner. For self-adjacent patches and blocks this is not necessarily true. On the implementation level, special care needs to be taken to ensure the correct handling of such cells. The ambiguity can concretely be resolved by using the additional geometric information available from the embedding into the background mesh. For instance, a block corner in the background mesh is clearly defined as a volume-connected sector of tets incident to a node-vertex, demarcated by embedded patches. Furthermore, one should deliberately avoid non-robust versions of path-finding or surface-finding algorithms; for instance, finding the shortest path between two corners of a patch might require explicit routing through the patch interior, if the corners are represented by the same node. In our definition of patch and block bisection (section 9.4.2), the correct homotopy of paths and surfaces is ensured through guaranteed-correct initialization and structure-preserving shifts.

9.5.2 Meta-Cell Geometry

While the effect of our collapses on the complex's connectivity is uniquely defined (cf. section 9.3.1), geometrically the embedding update comes with degrees of freedom. The embedded collapse operators in section 9.3.2.5 are designed to, in a sense, modify the embedding as little as possible. The embedding of nodes remains unchanged (unless deleted), the updated embedding of arcs and patches is effectively formed out of unions of their prior embedding and the collapsed cell's embedding—just pulled apart over one layer of background mesh elements to avoid non-injectivity. Depending on the use case, it can be desirable to afterwards further adjust this embedding, optimizing for some objective.

To improve the shape of the blocks after all collapses have been performed, we can optimize each arc's embedding for minimal length and each patch's embedding for minimal area—in the metric induced by the underlying parametrization based on which the motorcycle complex was built. This can be performed in a discrete incremental manner, using the ARCSHIFT and PATCHSHIFT operators to move arcs and patches, driven by these objectives. Alternatively, instead of proceeding incrementally, an arc's embedding can be directly replaced by a discrete shortest path and a patch's embedding by a discrete minimal surface [Grady 2008]. With the latter approach, ensuring that the resulting minimal surface is homotopy equivalent to the prior embedding image is a challenge. A selective fallback to the incremental approach then is convenient.

9.5.3 Background Mesh Refinement

Whenever an ARCSHIFT or a PATCHSHIFT would violate injectivity, i.e., two arcs or patches would become overlapping, refinement of the background mesh is needed. This creates the degrees of freedom necessary to validly maintain the embedding while performing the shift (effectively over part of the original triangle or tetrahedron). Specifically, an overlap may be either between the images of two distinct cells, or within one cell's image, making it non-manifold.

[1] https://github.com/HendrikBrueckler/C4HexMeshing

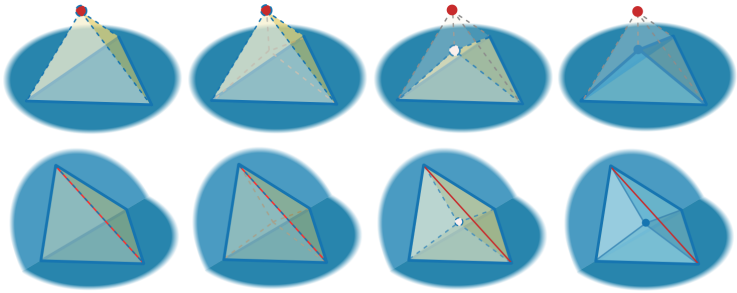


Figure 9.12: Background mesh refinement of a tetrahedron (1:4 split) to prevent overlap with a forbidden vertex or edge (red) in PATCHSHIFT.

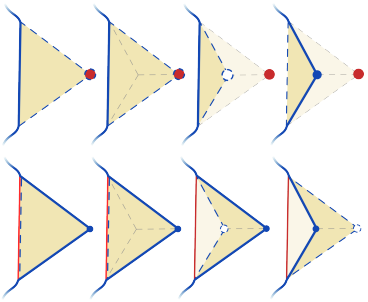


Figure 9.13: Background mesh refinement of a triangle (1:3 split) to prevent overlap with a forbidden vertex or edge (red) in ARCSHIFT.

In the following we refer to any background mesh element on which such an overlap would occur in the context of a concrete shift operation as *forbidden*. The following refinement rules are applied to prevent any kind of overlap:

I Before a standalone PATCHSHIFT: Split the tetrahedron (1:4) if it is incident to forbidden vertices, edges, or facets. Then the patch can be shifted across a subtetrahedron (fig. 9.12).

II Before a standalone ARCSHIFT: Split the triangle (1:3) if it is incident to forbidden vertices or edges. Then the arc can be shifted across a subtriangle (fig. 9.13).

III In ARCPATCHSHIFT, before a sequence of PATCHSHIFTS along a tet fan is performed: Split all inner triangles of the tet fan whose outer vertex or outer edge(s) are forbidden (cross-section view in fig. 9.15). Then recompute the fan.

IV In NODEARCPATCHSHIFT, before a sequence of ARCPATCHSHIFTS along a triangle fan is performed: Split any inner edges of the triangle fan whose outer vertex is forbidden (fig. 9.15). Then recompute the fan.

Note that for the latter two, it is necessary to consider the sequence of atomic shifts as a whole, and perform splits that are not the locally minimal splits (cf. fig. 9.14) that would be employed if each atomic operation was considered individually.

Between operations we remove vertices introduced by earlier refinement that are no longer needed for an injective embedding. This is done by collapsing an incident edge of the background mesh; as it undoes only previous refinement, we call this *de-refinement*.

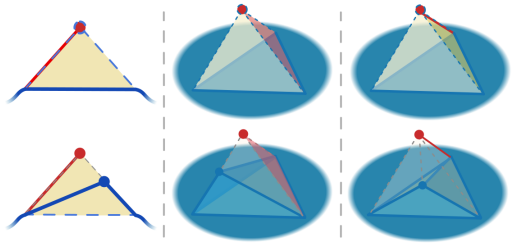


Figure 9.14: It can be necessary to employ non-minimal splits, i.e., edge splits for triangles (left) or tetrahedra (center), or facet splits for tetrahedra (right). Figure 9.15 shows a concrete scenario.

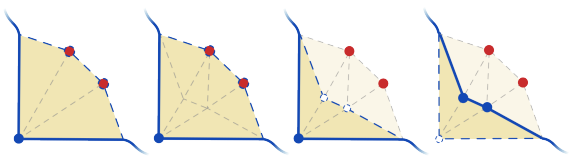


Figure 9.15: Background mesh refinement of a triangle fan (or a tet fan in cross section view) to prevent overlaps with forbidden vertices (red) during NODEARCPATCHSHIFT or ARCPATCHSHIFT. In the end, shifts can be carried out over subtriangles (or subtetrahedra) without causing overlaps.

9.5.4 Background Cell Geometry

Due to refinement of the background mesh (during embedding updates but also already during motorcycle complex tracing), the background mesh's quality can become low, containing elements with tiny angles. While not a problem for our collapsing method, it increases the hardness of the subsequent blockwise mapping.

To improve the numerical condition of the problem for mapping methods based on numerical optimization (which we prefer for their simplicity wherever possible), we remesh the background mesh while maintaining the embedding of the complex after collapsing has completed. For simplicity we make use of a straightforward greedy approach: The standard operations edge collapse, edge split, edge-face swap, and vertex shift (cf. section 2.2.2) are greedily applied on the background mesh wherever this locally improves the worst inner (facet and tet) angles (see section 2.2.3 for reasoning). To maintain the embedding, a collapse and a swap are only allowed *within* a set of background mesh elements that lie in the embedding image of the same block, patch, or arc, and only if this does not alter the geometry of critical entities of the background mesh.

9.5.5 Critical Entities

A little additional care is necessary when applying the collapses when some critical background entities (boundaries, features, or singularities) are to remain explicitly represented in the cell complex. The embedding of meta-cells covering such critical entities is crucial and must not be changed (except *within* the entity itself). In case of boundaries, this is strictly required to maintain embedding surjectivity. For atomic embedding shift operations, it is therefore important to shift nodes and arcs only within the critical entity that contains them; for a critical patch a shift is never applicable. For entire collapse operators there is a *direction* degree of freedom: Which of the two incident lower dimensional cells vanishes and which one remains (cf. section 9.3.1). When collapsing a cell where one of the two is marked as critical, we just always need to choose this one to remain, keeping its embedding unchanged. Note that, assuming a valid quantization, a (to-be-collapsed but incollapsible) zero-element between *two* marked elements (not within the same singularity or feature) will not occur due to the quantization preserving the structure of critical entities (theorem 7.3).

9.5.6 Last Successor

The operators NODEARCPATCHSHIFT and ARCPATCHSHIFT, in the form described above, first insert the traversed edge (facet) into the follow-up arc (patch), and then lift off the arc (patch) to prevent overlaps with subsequent ones. For the last arc (patch) encountered (cf. fig. 9.6p-r) this is not strictly necessary, since there are no further elements to be adjusted that could require the freed up space. As such, the shift operation should simply be skipped for the last successor—for efficiency and to aid the handling of boundary elements in the following. When shifts happen *along* a critical entity, it is furthermore crucial that successors are chosen in a sequence that preserves the relative embedding of both meta-cells *in* and *outside* of the critical entities. As argued before, in a valid quantization such a sequence must exist and can be found by performing (greedily) only feasible shifts.

9.6 Results

Datasets As input data for the purpose of testing and evaluation we use the two datasets mentioned in section 7.6.1. Furthermore, as these do not contain any explicit feature information, we create a third set of instances by computing and quantizing T-meshes on seamless parametrizations of models from the HexMe dataset [Beaufort et al. 2021], which comes with prescribed feature points, curves, and surfaces. These datasets then contain 15, 82, and 110 instances, respectively, in total: 207.

On these, non-negative quantizations of varying target resolution are computed. In particular, to maximally challenge our method, for the stress test in section 9.6.1.1 for each instance we compute a maximally coarse quantization. This effectively maximizes the number of zero-elements, and—consequently—the collapsing and embedding update effort for our method. As it also implies the largest deviation between an initial seamless map and the to be computed IGM, it also serves as an ideal basis for gauging the performance of our blockwise remapping approach compared to previous global remapping methods. For more detailed comparisons in section 9.6.2.2 we furthermore generate sequences of increasingly fine quantizations. Finally, for the purpose of generating block-structured hexahedral meshes in section 9.6.2.4, we generate a moderately coarse quantization to determine a *block layout*, which is potentially scaled up to yield a sufficiently fine quantization in the end (with the same block layout).

9.6.1 T-Mesh Collapsing

We evaluate the performance of the suggested collapse operators for T-meshes using our own reference implementation (section 9.5), applied to T-meshes constructed and quantized via the methods from previous chapters.

9.6.1.1 Stress Test

In 168 (81%) of the test instances, the stress test quantization contains zero-elements. As nothing is to be done on the rest, we restrict ourselves to these in the following. To give an idea, in these instances 25% of blocks are zero-blocks, 23% of patches are zero-patches, and 15% of arcs are zero-arcs.

Our operators, applied using the strategy described in section 9.4.3, successfully collapse all zero-arcs, zero-patches, and zero-blocks in all of these instances. The total number of operations applied over all instances is 55,687. Concretely:

- 19,167 arc collapses,
- 22,912 pillow-patch collapses,

Table 9.1: For inputs from the first dataset, absolute numbers and percentages of zero-elements (blocks, patches, arcs) in the coarsest quantization are shown in grey, followed by the overall percentage of zero-elements. Furthermore the number of collapses (block, patch, and arc) and bisections (block and patch) performed to remove all zero-elements are shown, followed by the grand total of operations and the overall reduction of cell count. Models for which the coarsest quantization contains no zeros are omitted.

Model	$\lvert B_0\rvert$		$\lvert P_0\rvert$		$\lvert A_0\rvert$		$\frac{\lvert S_0\rvert}{\lvert S\rvert}$	$\updownarrow_B$	$\updownarrow_P$	$\updownarrow_A$	$\lvert_B$	$\lvert_P$	Σ	$\Delta\lvert S\rvert$
ARMADILLO	23	8\%	120	10\%	122	8\%	8\%	35	144	130	12	23	344	16\%
CAMILLE HAND	14	20\%	50	18\%	40	11\%	15\%	24	68	48	10	16	166	25\%
JOINT	11	32\%	46	28\%	44	18\%	23\%	12	51	46	1	5	115	35\%
SCULPTURE	13	48\%	52	41\%	52	28\%	34\%	13	50	48	0	0	111	52\%
ROCKERARM	5	4\%	38	6\%	48	6\%	6\%	9	44	49	4	2	108	10\%
FANDISK	5	18\%	25	19\%	27	13\%	15\%	8	31	29	2	6	76	29\%
CYLINDER	4	44\%	16	38\%	16	25\%	31\%	4	16	16	0	0	36	49\%
BONE	1	2\%	8	4\%	8	3\%	4\%	3	10	8	2	0	23	7\%
KITTEN	2	4\%	7	3\%	6	2\%	3\%	2	7	6	0	0	15	4\%
BROKEN BULLET	0	0\%	2	3\%	4	4\%	3\%	0	2	4	0	0	6	5\%

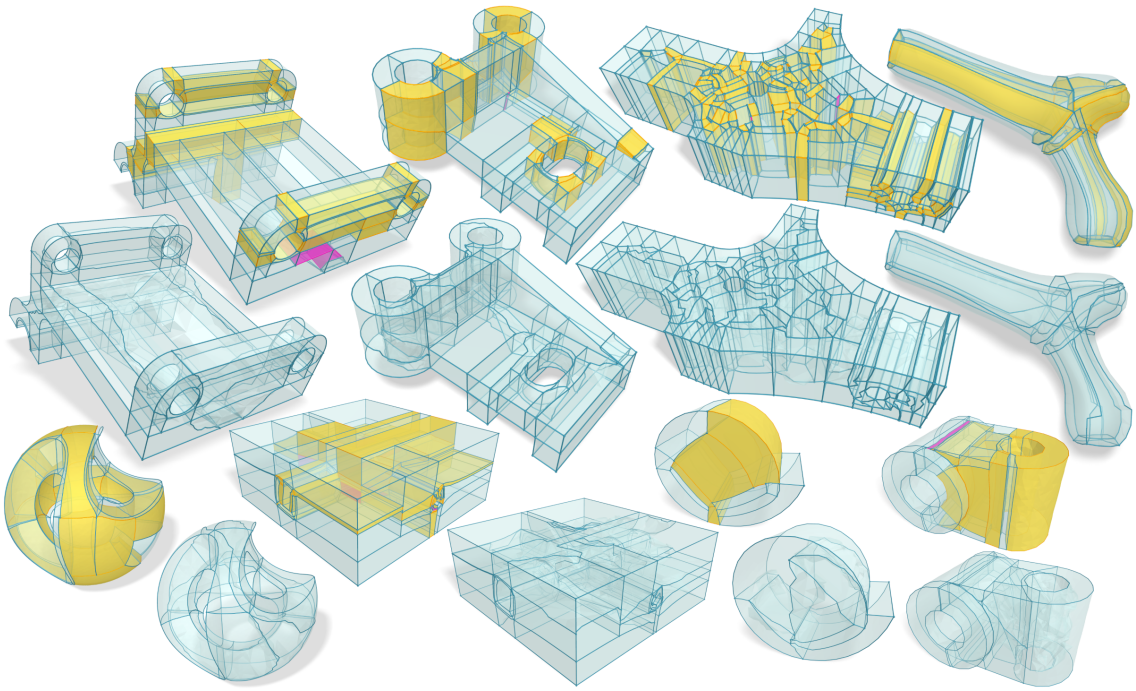


Figure 9.16: Several models with embedded T-mesh complex, before and after performing our collapsing routine. Collapsing was performed in accordance to the maximally coarse quantization, eliminating all zero-arcs (orange), zero-patches (purple, if isolated), and zero-blocks (yellow) in the process. The embeddings of arcs and patches affected in the process were geometrically relaxed in the end (section 9.5.2). Video animations of the process are available in the supplement of [Brückler and Campen 2023].

- 6,539 pillow-block collapses,
- 4,866 patch bisections,
- 2,203 block bisections.

The pillow-block collapses include 23 collapses of blocks with only one incident patch. On average, the total number of cells in the complex reduces by 29% in the process. For details regarding these numbers, on a per instance basis, refer to [Brückler and Campen 2023, Tab. 1–3].

Validity of the result was algorithmically verified in each case, checking surjectivity, injectivity, and manifoldness of the embedding. Validity of the quantization on the resulting reduced complexes was verified in all cases as well, all patches are of rectangular type and all blocks of cuboidal type, in terms of connectivity as well as geometrically under the quantization. No elements of length, area, or volume zero (under the quantization) remain in any case.

Figure 9.16 shows several of these instances, before and after the collapsing procedure. Let us point out again that we do not mean to advocate the use of these *maximally* coarse quantizations, that we use for stress testing, for practical purposes in general. As can be seen in the example in fig. 9.23, such a maximally coarse result can be very distorted (mainly because the fixed predetermined singularities are not well-placed for a structure this coarse). Regardless, our method handles it properly.

Runtime The average processing time of our implementation on these datasets—for the maximally coarse quantization, which implies the highest amount of collapsing effort—is 308s, with a median of 109s. Of all instances, 88% finish in under 10min. While not quick, note that this is of the same order as T-mesh construction and quantization computation. An outlier among the 168 instances takes 95 min. This is due to the background meshes taken as input having sizes up to 2.7 million tetrahedra, with cell complexes up to 8787 cells. There certainly is room

for improvement. For instance, large parts of the runtime (around 80%) are spent on adjusting (refining, de-refining) the background mesh in our reference implementation based on a data structure not ideally suited for this dynamic adaptation.

To reduce the hardness of the blockwise mapping (discussed below), we add a geometric embedding optimization and background mesh remeshing (see section 9.5.4) to this pure collapsing method. With our current simplistic implementation of this, runtime including this remeshing is increased to 19.8 min on average, with a median of 3.7 min. This optional ingredient allows computing the blockwise maps almost entirely with relatively simple optimization based methods (as discussed below). Without it, escalation to a more complex guaranteeing mapping method, by contrast, would be necessary in around 20 times as many cases in our stress test scenario.

9.6.2 Guaranteed-Injective Integer-Grid maps

On the T-meshes obtained from the collapsing evaluation, we compute block-wise integer-grid maps with guaranteed injectivity, stitched together compatibly across block boundaries. These are used to initialize a global continuous integer-grid map optimization (resulting in correct integers by using the constraints from chapter 8), from which hex meshes are extracted. This process is detailed in the following.

9.6.2.1 Block-Wise Integer-Grid Map Construction

After collapsing all zero-elements of the complex (and optimizing the embedding as discussed in section 9.5.2), we are left with a complex such that for each block $b \in B$ the associated quantization defines a positive extent $m_b \times n_b \times o_b \in \mathbb{Z}^3$. Note, that the simple re-scaling strategy from section 6.4.1 is not applicable, as after collapsing the T-mesh is no longer aligned with the original seamless map. However, because the labels of the T-mesh are preserved throughout the collapsing routine though, each patch and block still unambiguously represents an axis-aligned parametric rectangle or cuboid, respectively.

This leads to a mapping problem per block, computing a map $\phi_b : \mathcal{I}(b) \to D_b$, i.e., from the portion of the background mesh $\mathcal{M}$ which b is embedded in, to an axis-aligned origin-rooted cuboid $D_b = [0, m_b] \times [0, n_b] \times [0, o_b]$ of the specified size. The boundary map $\phi_{\partial b} : \mathcal{I}(\partial b) \to \partial D_b$ needs to be prescribed to ensure compatibility with adjacent blocks. Concretely, note that the block's boundary is covered by patches and arcs. Each patch p has a target extent $m_p \times n_p$ assigned by the quantization, and each arc has length q_a. The boundary map $\phi_{\partial b}$ is chosen such that each such patch maps onto an axis-aligned rectangle of size $m_p \times n_p$ in ∂D_b and each arc onto an axis-aligned interval of length q_a in ∂D_b. The interior of a patch can be mapped into this rectangle arbitrarily; this just needs to be done the same way (up to integer translation and transitional rotations) for both blocks that are incident on the patch; equivalently, patch-wise maps must agree on arcs. The union of these maps ϕ_b defines a global IGM ϕ_q for $\mathcal{M}$, namely one in which each block forms a chart $(\mathcal{I}(b), \phi_b)$ and transitions between charts can be inferred from the two block-wise maps per patch. These patch-associated transitions, due to patches being mapped compatibly and block-wise maps differing by exactly an integer translation and a (signed) axis permutation, are exactly integer-grid automorphisms theorem 3.1 required for an IGM.

Such a mapping problem, from a ball-topology region to a convex (specifically cuboid) region, without cuts, transitions, singularities, overlaps, and with fixed boundary, is more standard and arguably simpler than a global volumetric parametrization problem with free boundary, constrained transitions, interior pin constraints, etc. The recent method of Hinderink and Campen [2023] provides a reliable solution, supporting boundary constraints and guaranteeing bijectivity.

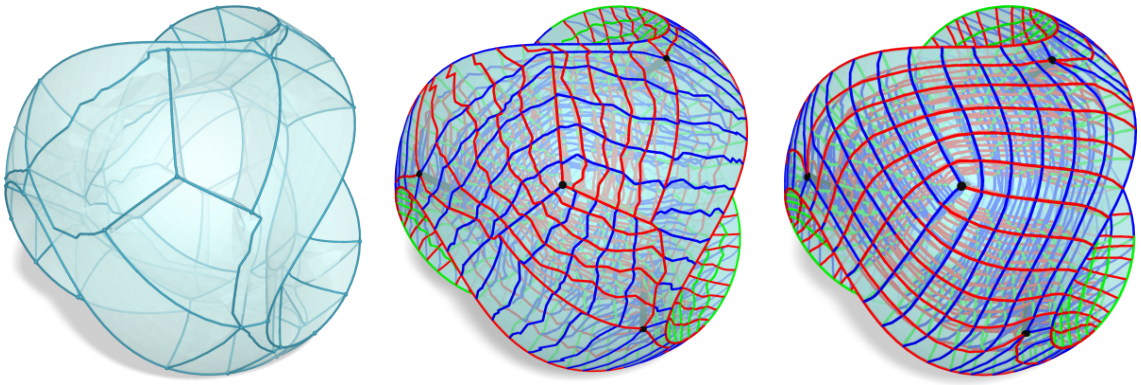


Figure 9.17: While collapsing may result in some blocks being geometrically distorted or jagged (left), causing an initially distorted IGM after blockwise construction (center), a subsequent global map optimization starting from this valid initialization reduces map distortion (right)

Several other recent optimization-based methods [Du et al. 2020; Garanzha et al. 2021], while not guaranteeing bijectivity, show high success rates. We combine their strengths for our evaluation purposes, first using an optimization-based method for its efficiency and escalating to the guaranteed method if necessary. In this way we close the robustness gap due to step (3) in the hexahedral mesh generation pipeline.

The constructed global map can finally be optimized for reduced distortion, while maintaining local injectivity and the quantization. We employ a symmetric Dirichlet objective with flip preventing line search [Rabinovich et al. 2017] and linear constraints that preserve seamlessness and keep the integer values at singularities, features, and boundaries fixed (cf. fig. 9.17). Due to this optimization, across block boundaries, the concrete geometric embedding of the collapsed complex (used merely for initialization) is of low relevance. Specifically, the integer-fixing constraints are constructed by the routine from chapter 8. Note that the initial block-wise IGM trivially satisfies these constraints: The prescribed integer displacements are obtained from vectorial sums of arc lengths, and because each arc is parametrized exactly to its quantized length and retains its orientation, sums must match as well.

Global statistics We computed blockwise maps for all 10,433 blocks of the collapsed complexes using the mapping method from [Du et al. 2020], followed by the method from [Garanzha et al. 2021] if it does not succeed. For these the block boundary maps were computed per patch as 2D Tutte embeddings, and the interior map initialized as a 3D Tutte embedding prior to further optimization. For 99.96% of all blocks, the map ϕ_b resulting from numerical optimization was fully bijective. The remaining 0.04% (4 blocks total from two different input models) have a few remaining inversions each with this strategy. By applying the reliable, combinatorial method of [Hinderink and Campen 2023], also for these a bijective map was obtained.

9.6.2.2 Comparison

Our approach for constrained global IGM recomputation from scratch (chapter 8), using global non-convex optimization, has been mentioned and used also by Liu and Bommes [2023, Sec. 6.2] in their work on *Locally Meshable Frame Fields*, using a different combination of objectives, constraints, and solvers. We refer to our own global recomputation routine from the previous chapter (without injective initialization) as GLOB-OPT and to the one from [Liu and Bommes 2023] as LMFF-OPT—referring to just this optimization part, not the methods as a whole.

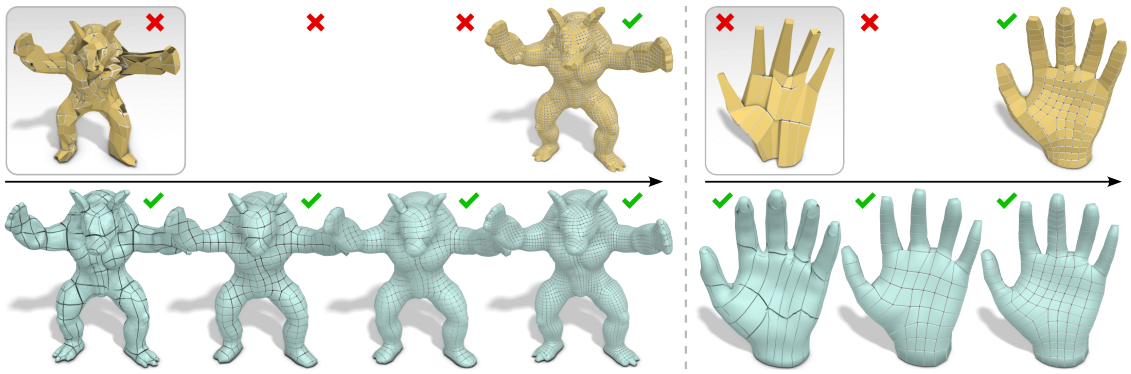


Figure 9.18: Two examples of input instances (ARMADILLO and CAMILLE_HAND) demonstrating the unconditional robustness of our method (bottom row) in comparison to GLOB-OPT (top row). The coarsest versions are shown with non-linear curved elements for visual clarity. GLOB-OPT fails to obtain a valid IGM in all but the rightmost case for both models. We can still attempt to extract a hexahedral mesh, but even a fault-tolerant extractor [Lyon et al. 2016] only yields a partial mesh (shown in the insets) with holes and other defects.

Figure 9.19: To highlight the significance of robustness in IGM reparametrization, the failure rates of a state-of-the-art global reparametrization scheme LMFF-OPT applied to the 110 inputs from the third dataset are evaluated as a function of target complexity. Note that 'target #hexes' is what the quantization aims for; what ultimately results is of course limited by the prescribed singularity structure.

GLOB-OPT As discussed in section 8.4 the global recomputation of a quantized IGM from scratch is non-robust due to potential inability of optimization routine to find an injective map, especially for coarse settings. This is relieved by the method presented in this chapter, as it guarantees a constraint-feasible and injective initialization of the optimization. As an example, for the first two models from table 9.1, as shown in fig. 9.18, using the method presented in this chapter, we are able to compute valid IGMs for maximally coarse quantizations (as reported in section 9.6.1.1), whereas GLOB-OPT succeeds in achieving a valid result only when the target resolution is increased (using a trial-and-error search approach) so much that the resulting meshes have 22× and 10× as many hexahedra, respectively.

LMFF-OPT Liu and Bommes [2023] report that LMFF-OPT (nearly) always succeeds in their experiments when targeting dense resolutions of 100,000 hexahedra, but that for "extremely coarse quantizations ... failures can be observed." We confirm this using the authors' released implementation: Figure 9.19 shows how the success rate of LMFF-OPT decreases with a decreasing target mesh resolution. For a maximally coarse quantization, it fails to obtain a valid IGM in 44%

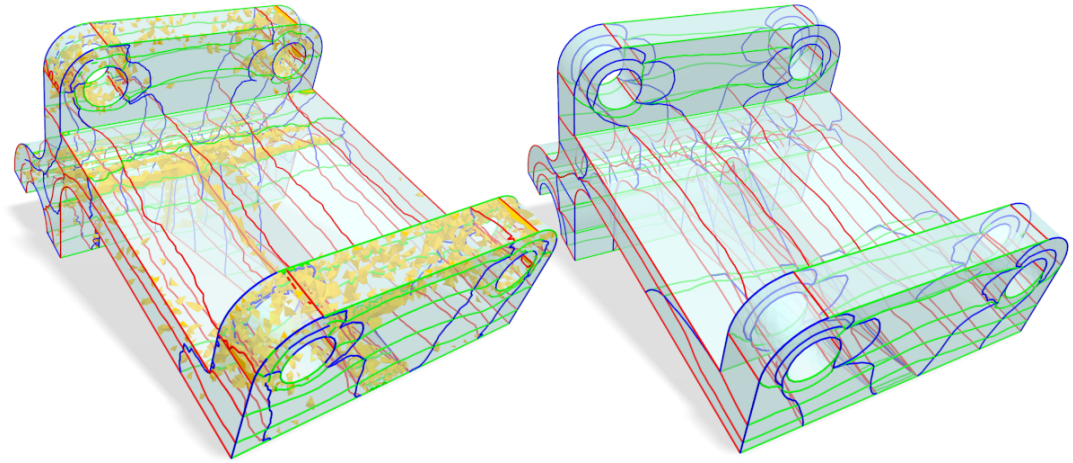


Figure 9.20: Example (I06U M6) from the third dataset for which a previous approach, LMFF-OPT, yields a map (left) that inverts many tetrahedra (highlighted in orange) for a maximally coarse quantization setting. Our method obtains a valid IGM (right). Shown are the parametric integer-grid isocurves, colored according to local U- (red), V- (green) or W-alignment (blue).

of the cases. Figure 9.20 shows an example. For somewhat finer target resolution, the success rate quickly rises above 90%, but a small gap remains even when increasing to a very high resolution of 500,000 hexahedra. This gap is reliably closed by the method presented in this chapter.

Note that the resulting hex mesh connectivity is identical, regardless of method (if successful), as we are not comparing different singularity structure determination methods or different quantization methods here; the target IGM is the same in all cases. The contribution presented in this chapter should therefore not be misunderstood as aiming for a different mesh of potentially higher quality. Instead, the key benefit is the unconditional robustness it ensures, whereas previous methods fail unpredictably if the target resolution parameter is not set high enough. Of course, even with identical mesh connectivity, there may be quality differences in terms of distortion, but this is mainly a question of the applied final map or mesh optimization. We investigated scaled Jacobian values, assuming simple linear hexes, in comparison to LMFF-OPT; a few differences can be observed (as can be expected due to different map optimization paths, starting points, etc), but no systematic advantage to either side.

9.6.2.3 Feature Preservation

The third dataset, based on instances from the HexMe dataset, comes with feature annotations. Some vertices, edges, or faces in the underlying tetrahedral mesh are marked as (part of) a feature point, curve, or surface. Because we have carefully tuned the various stages of the T-mesh generation and quantization computation to respect these features, and designed our collapse strategy to likewise take them into account, the generated map ϕ, and thus an implied hexahedral mesh, is aligned to these features. In particular, chains of hex edges follow the feature curves, sheets of hex faces cover the feature surfaces. Figure 9.21 demonstrates this on an example from the dataset (the same model displayed in fig. 5.9, right).

9.6.2.4 Block Structuring

Coarse IGMs are not only of academic interest (e.g., for stress tests as in section 9.6.1.1). While maximal coarseness can imply excessive distortion (fig. 9.23), moderately coarse IGMs can be used to determine and define *block layouts* for block-structured (or multi-block) hex meshes, which are

of practical interest [Kopriva 2009; Armstrong et al. 2015]. This is similar to how *quad layouts* are used for the generation of patch-structured quad meshes [Campen 2017; Lyon et al. 2021a].

To demonstrate this, we take a moderately coarse quantization and perform our collapsing routine accordingly on the motorcycle complex. The result essentially determines the block layout. A finer (strictly positive) quantization, with a scale chosen based on the desired hex mesh resolution, is then computed *on this reduced complex*. The implied IGM can then be constructed in a blockwise manner and optimized (as illustrated in fig. 9.17). Finally, a hex mesh can be extracted. fig. 9.22 shows example results of this process; it visualizes the meshes' *base complex*, serving as a gauge for the mesh's regularity (cf. section 2.3.3.2). A simple base complex with few blocks is generally preferred, because it signals a highly regular mesh, preferrable due to potential

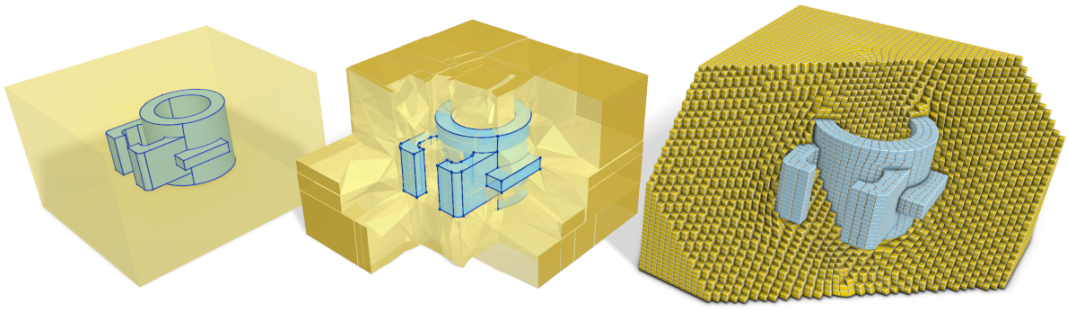


Figure 9.21: Some input instances (like I15B S8, left) contain marked feature points, curves (dark blue), or surfaces (light blue) in the volume, that the result is supposed to respect. Our motorcycle complex construction routine properly aligns the T-mesh complex to these. Preserving feature nodes, arcs, and patches during collapsing (center) guarantees that the resulting hexahedral mesh is aligned with these features (right). Some elements have been peeled away in this visualization to reveal the interior, and hex mesh faces that are aligned with a feature surface are colored blue.

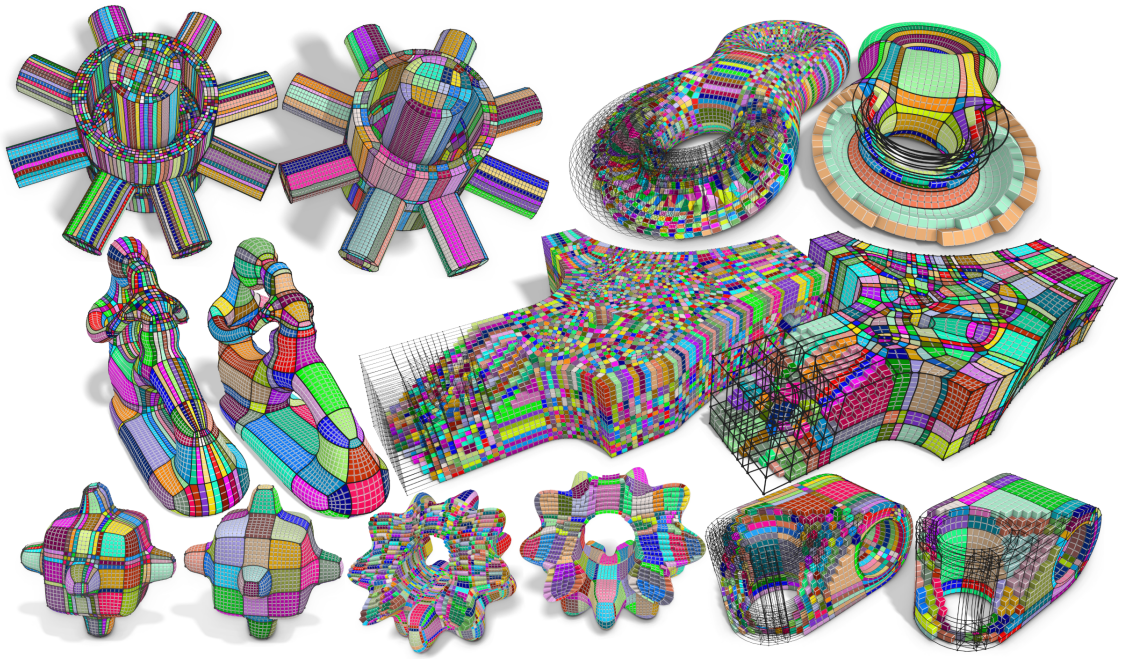


Figure 9.22: Pairs of hexahedral meshes (of comparable resolution) with hexahedra colored according to blocks of the mesh's base complex. Some are sliced to reveal the interior. The right mesh of each pair (with a significantly simpler block structure) is the result obtained when imposing a coarse block layout as described in section 9.6.2.4. For comparison, the left mesh of each pair is the result obtained when directly targeting the final mesh resolution, without the use of our collapsing capabilities to impose a block structure or eliminate zero-elements.

performance benefits for applications. Note that even coarser block layouts can be imagined for some of these models. This would be a matter of using a different singularity structure, though, which is a fixed input in our current scenario.

9.7 Discussion

Operators We have introduced a set of operators to perform collapses in volumetric cell complexes that are discretely embedded in a background mesh. Conforming complexes as well as non-conforming cuboidal complex are supported. Applying these to T-mesh complexes used in the context of hexahedral mesh generation allowed us to reduce a hard step (constrained seamless map computation) to a set of simpler problems (convex simple map computations). For these there are reliable solutions available due to recent advances [Hinderink and Campen 2023; Nigolian et al. 2023].

In fact, the embedding-altering operators introduced here have been used in an extension of the reliable method from [Hinderink and Campen 2023], which targets star-shaped parametric domains, to one that reliably maps to arbitrarily shaped domains of ball topology [Hinderink et al. 2024]. At the core of the extended method, a partition of source and target domain has to be found, such that the latter parts are star-shaped and the partition is of equivalent topology in both domains. To guarantee exact topology control, an approach similar to ours is used, adjusting the partition embedding incrementally from a known-valid initial state using a variation of patch shift operators introduced here.

We hope and expect that the introduced operators can be of value for further use cases and application scenarios, especially as volumetric geometry processing can be witnessed to be of increasing interest and relevance.

With regard to our embedding-maintaining volumetric cell complex operators, the amount of mesh refinement to enable the embedding updates can be quite high in unfavourable configurations and constitutes a performance bottleneck. It will be interesting to explore ways to determine the updated embedding images of the individual cells not with a focus on minimal geometric change, but with a focus on minimal required refinement.

Fully robust quantization The collapse operators introduced in this chapter, together with the application of guaranteed-injective mapping schemes on the remaining T-mesh blocks, constitute the last missing piece to fully close the robustness gaps between a continuous seamless parametrization and a quantized integer-grid map. Let us shortly recapitulate the pipeline outlined in sections 1.5.1 and 3.7 after inserting our robust algorithms:

I Tet mesh the domain.

II Compute a meshable frame field on the tet mesh, fixing rotations of transitions.

III Compute a continuous seamless map with fixed rotations, without integrality constraints.

IV Quantize the seamless map into an integer-grid map.

- **IVa** Correct numerical issues in the seamless map, making it *truly seamless* .
- **IVb** Construct a coarse T-mesh decomposition, via the *3D motorcycle complex* algorithm.
- **IVc** Compute a valid proxy T-mesh quantization by assigning integer lengths to it.
- **IVd** Extract linear constraints from the quantized T-mesh.

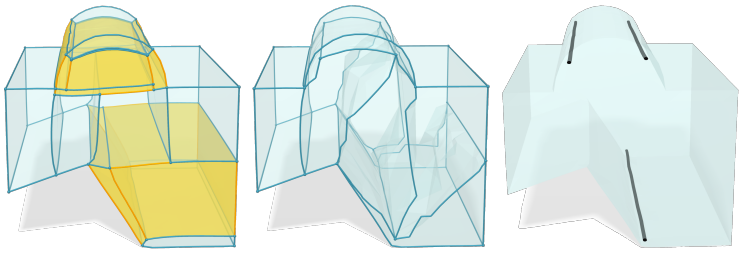


Figure 9.23: Example of a block structure (left: input, middle: collapsed) resulting from a maximally coarse quantization, with low geometric quality—in particular due to fixed singularity positions (right).

IVe Collapse zero-cells from the T-mesh.

IVf Initialize a quantized, injective map per block; optimize it globally, s.t. **IVd** .

V Extract a hex mesh by pullback of the integer grid along the integer-grid map.

The quantization steps under **IV** are indeed quite a complex machinery for the seemingly simple problem of 'just fixing the integers'. In a practical pipeline it might be advisable to employ cheaper heuristic approaches first, and resort to the equivalent robust method only if these fail. Nonetheless, we have demonstrated that heuristics fail regularly in practice and that a lack of robust fallback impairs the usability of the entire pipeline.

In the following chapters we focus on aspects that improve the usability, efficiency and flexibility of this pipeline, targeting once more the quantization stage. Concretely, **IVc** involves solving (pure) integer quadratic programs. Of course general-purpose branch-and-cut solvers can be employed for the solution of these, but these come with problems: Hard to predict performance (exponential in the worst case), as well as potentially costly licensing and closed-source code. In chapter 11 we develop a special-purpose solver that exploits the special structure of the T-mesh quantization problem.

Furthermore, we observe that a suboptimal singularity structure adopted from stages II and III can impair result quality significantly, either by causing many irregular edges and a dense base complex (see fig. 9.22, center right), or by imposing limits on geometric quality (especially for coarse targets, see fig. 9.23) due to being handled as immovable fix-points in the pipeline. We therefore explore variations of **IVc** and **IVe** , that no longer treat singularities as critical. This allows on-demand simplification of the singular graph by collapsing singularities and recombining them into regular edges, executed as part of the collapsing process of, but requires a more principled approach to still ensure quantization validity (which was previously upheld partially by the singular graph, see section 7.3.4).

Part D

Towards Versatility and Efficiency

There is surely nothing quite so useless as doing with great efficiency what should not be done at all.

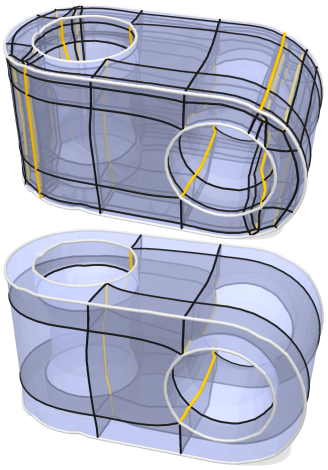
Peter Drucker

10 Flexible Singularities

IV Quantize the seamless map ϕ into an integer-grid map ϕ_q.

- IVa–b Embed a ϕ-aligned T-mesh $\mathcal{T}$ in the volume
- **IVc** Quantize $\mathcal{T}$ into $\mathcal{T}_q$, relax singularities but preserve features
- IVd Extract constraints for ϕ_q
- **IVe** Collapse zero-cells from $\mathcal{T}_q$, removing some singularities
- IVf Compute valid ϕ_q per cell, then optimize s.t. constraints

V Extract a hex mesh from ϕ_q.



Notably, all recent volumetric methods based on the general integer-grid hex meshing pipeline, including our methods from the previous chapters, assume the singularity structure determined via frame-fields to be fixed throughout all following steps. This can pose a suboptimal restriction, as the frame field cannot properly take into account the discrete output mesh resolution determined in later steps, even though this aspect is strongly coupled to the singular structure. Aside from that, frame field design is still a problem with many limitations; meshability is the primary hindrance, but even meshable fields might yield overly messy singular graphs. An overly complex singular structure in the early stages will thus also result in a more irregular hexahedral mesh while meshes with sparse and well aligned irregular edges are often preferred in applications (cf. section 2.3.3.2).

In the surface setting, concerning quadrilateral mesh generation, allowing changes (in particular simplification) of the singular structure within the quantization step section 1.5.2 has already proven instrumental in achieving highest-quality results [Lyon et al. 2021b]. With the same goal in mind we propose a volumetric quantization method that incorporates the possibility of simplifying the singular structure, not as a heuristic pre-process or post-process, but as integral part of the quantization step chapter 10. The simplification is steered directly by the minimization of a distortion measure that aims to preserve scale and spacing of mesh features between the continuous and quantized seamless map, proxied once more via T-mesh quantization.

Related Work: Singularity Modification There are several methods that describe modifications to the singular structure not as a part of quantization, but rather as a repair measure to

ensure meshability, i.e., the existence of a valid parametrization and corresponding quadrilateral or hexahedral mesh. On surfaces these measures usually only encompass the addition of a few singularities where necessary [Myles et al. 2014; Pietroni et al. 2016, 2021], while volumetric meshability is a much more complex issue requiring a whole suite of operators that e.g., merge, zip or unzip singular curves [Jiang et al. 2014; Fang et al. 2016; Liu et al. 2018; Reberol et al. 2019; Liu and Bommes 2023]. In a more theoretical endeavor, it was demonstrated how any hexahedral singular node, the branching point of singular curves, can be completely decomposed into non-branching singular curves [Zhang et al. 2023]. More distantly related are works modifying hexahedral meshes by post-processing rather than during generation, often with the goal of simplifying their intrinsic block layout, i.e., their base complex. Such methods modify primal sheets of the hexahedral mesh [Borden et al. 2002; Ledoux and Shepherd 2010; Gao et al. 2015a], dual chords [Tautges and Knoop 2003; Kowalski et al. 2012] or both [Gao et al. 2017b], the latter also allowing singularities to be merged. For these methods, a parametrization on the original domain is no longer present, making the preservation of features or the boundary shape difficult; hence, the structure simplification of such methods is often coupled to a loss of detail. Compensating for this in advance would require strategies that generate overly fine meshes in the first place, something that structure simplification methods working on the quantization level can avoid completely.

10.1 New Problem Statement

The high-level goals pursued in this chapter can be summarized as follows: Relax the restriction of previous quantization methods that the singularity structure must remain fixed, and instead require only that explicit features of the input must be exactly preserved. Furthermore, measure the accuracy of the T-mesh quantization (with respect to parametric target lengths from the input map) not by reproducing per-arc targets with per-arc integers, but by reproducing overall continuous sizes and spacings of features with sums of per-arc integers. Lastly, apply simplifications to the singularity structure implicitly within the T-mesh quantization when this benefits quantization accuracy. Simplifying the singular graph is possible either by collapse of entire singularities, or by merging different singularities: The resulting merged singularity has an index equal to the sum of its precursors, and ideally becomes regular in case of a pair of singularities with indices -1 and 1 merging.

Primarily, this approach decouples the target hex mesh density from that of the T-mesh (which in turn is inherently linked to the singular graph density), and allows to adapt the complexity of the singular graph to better match the desired hex mesh resolution. Beside a better matching of hex mesh resolution, this additional flexibility is also useful to tune the simplicity of the block layout for block-structured meshing. First, simplify the block layout, including the singularity structure, to a desirable level via a coarse quantization; second, compute a finer quantization on the simplified layout to obtain a hex mesh at the desired resolution.

In the following we list the concrete differences of the flexible-singularity approach with respect to the settings discussed in previous chapters.

Objective Function For volumetric quantizations, only $d(q, \ell) = \sum_i (q_i - \ell_i)^2$ has been employed as an objective function so far, i.e., a least-squares matching of quantized and continuous (target) arc lengths. However, despite the arc-based problem formulation, effectively only the integer coordinates of critical entities are handled as fixed points of the integer-grid map in the later quantized reparametrization, meaning only the total accumulated integer spacings in between matter. Individual integer arc lengths only determine how the map is initialized locally;

any non-critical map coordinates are allowed to change in a later map optimization step, as shown in fig. 9.17. Section 10.3 introduces a more global, feature based quantization objective that measures deviation of sizes and spacings between the actually relevant features, not of lengths of the auxiliary arc variables. This allows individual arc lengths to move further away from their individual target lengths as long as that better preserves feature shapes and spacings.

Critical entities In previous chapters, the term critical entities was used to mean both singularities and user-defined features. It was required that both should be aligned with the seamless map and should lie on integer coordinates of the later integer-grid map. Additionally, it was demanded that neither may move (in object space), disappear or merge. For the here discussed setting we adjust this notion: We keep the map-related requirements for both types of critical entities—this part is necessary to obtain a feature-preserving hexahedral mesh—but we demand only features to not move, disappear or merge; singularities are not prevented from doing so.

This means, for the T-mesh quantization and collapsing we consider singularities as non-critical while following the same goals formulated in previous sections with respect to such entities; or, equivalently, we narrow down the goals to include only feature entities rather than all critical entities. Both have the exact same effect; in the following we assume the latter view, and narrow down the goals of T-mesh quantization and collapsing to apply to feature nodes, arcs and patches (the latter grouped into feature links and regions analogously to theorem 7.5). We furthermore enforce that boundaries are always feature-marked, such that their correct treatment is covered by that of features. Of course, a user can still choose to fixate also a subset of singularities, by explicitly tagging them as features.

Structure preservation Within the T-mesh quantization, beside any user-defined features we also explicitly preserve implicit (topological) features such as external boundaries, internal boundaries (voids) and handles (see section 10.2.3), whose preservation was previously implied by the preservation of singularities. The structure-preservation requirements from section 7.3 can then be formulated as:

i) Two points on feature entities (distinct or connected via a self-loop) must not be contracted, except along paths lying *completely within* the union of the lowest-dimensional feature entities containing them.

ii) Cycles and spheres that are topologically non-contractible within $\mathcal{T}$ or within one of its feature entities (e.g., within the boundary) must not be contracted.

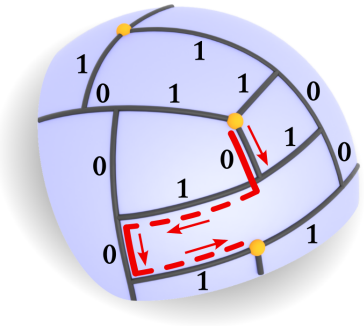
Note that this is a more concise version of previous requirements: Rather than explicitly demanding that individual feature entities $c \subset \mathcal{T}$ do not collapse to points or curves (as formulated previously), we recognize here that requiring the topology of each restriction $\mathcal{T}|_c$ to not change has the same effect. This is ensured by preserving interior topological features of each c via (ii) (i.e., keeping the first and second homotopy group $\pi_1(\mathcal{T}|_c)$ and $\pi_2(\mathcal{T}|_c)$ invariant under the quantization), and by recursively also preserving and separating each of c's boundary components via (i) and (ii). The requirements extend to boundaries of features, because any feature entity's boundary is by construction represented as additional lower dimensional features. In this way, all feature points, curves and surfaces are covered. Lastly, the topology of the 'volume feature', represented by the entirety of $\mathcal{T}$, is also preserved via (ii) (and recursively for its boundary $\partial\mathcal{M}$).

Due to the tight connection between topology of the domain and the singular structure of any boundary-aligned seamless map on it, in all but a specific edge case (a hollow torus) the network of singular curves necessarily spans the topological features (handles, cavities) of the volume. Hence,

maintaining the singular structure and domain boundaries, indirectly also preserved the domain topology without any additional care needed (cf. section 7.3.4). Our previous method—which assumed all singularities to be fixed (i.e., to-be-preserved features)—only had to actively prevent collapse or merging of critical entities in general.

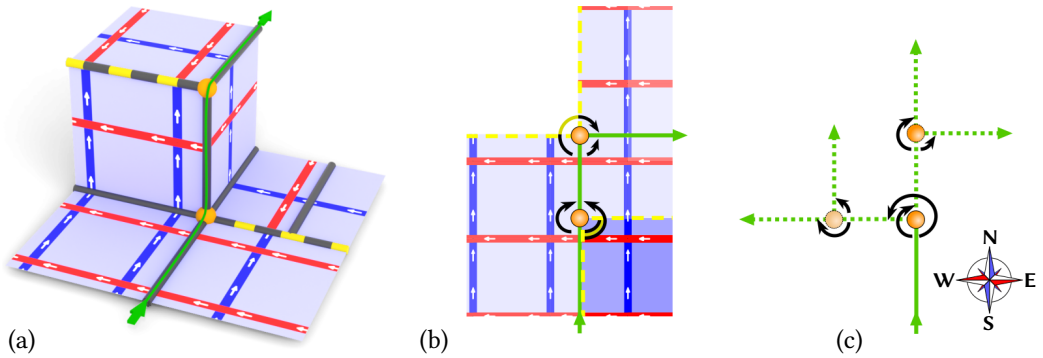
This was done using simple constraints that ensure that the length of critical curves is ≥ 1, together with additional, more involved *separation* constraints. Because explicitly separating each pair of critical entities would introduce a lot of unnecessary overhead, it proved expedient to derive and add only the separation constraints proven necessary in a lazy manner (cf. section 7.4). This is done by repeatedly solving the problem, and only explicitly adding constraints that are proven to be violated by the previous solution, until no more violations are found. We adopt this lazy approach, but a replacement of both the violation detection routine as well as to the constraint formulation is necessary.

Violation detection and constraint formulation Detection of violation was previously carried out via a graph search on the T-Mesh structure, starting from each critical entity and registering the shortest path that implies overlap with another critical entity (or with itself along a topologically non-trivial cycle) under the quantization. This path is a sequence of arcs; constraining a sum over these to be greater than 0 will enforce separation of the involved entities in the next solution iteration. If all singularities are considered features, all paths considered within the graph search are by construction always confined to regions of the seamless map that lie *between* singularities but never expand past these. Such regions are regular, so parametric directions of arcs are unambiguous. Therefore, notions like path segments being *parallel*, *antiparallel* or *perpendicular* have been used to determine signs for the sum constraints that ensure separation. In that specific setting, separation constraints achieve separation in a single step over-restricting the solution space only minimally, by requiring the relative alignment of singularities. Notably, this does not translate to the current setting, where singularities are no longer fixed features and paths may extend beyond (and wind around) these on their way to the nearest actual feature.



This issue of relative directional ambiguity of path segments when the path winds around singularities is illustrated in fig. 10.1 on a 2D example. Concretely, whenever the path crosses a 2D singularity, choosing whether to wind around it in clockwise or counter-clockwise order creates a bifurcation in the possible labeling of the remaining segments. Allowing the path to tightly wind around the same singularity more than once would increase the branching factor even further, as do 3D singular nodes. Technically, because a volumetric seamless map is equivalent to a scalar-valued function on a 24-sheeted branched covering [Nieser et al. 2011], up to 24 directions would be needed to unambiguously label a 3D path globally.

In section 10.2 we describe a novel method to generate structure-preserving constraints that separately and explicitly prevent changes to features *and* to domain topology, both without relying on a fixed singular structure. This requires tracing truly global paths within the T-mesh, i.e., ones that may pass by or wind around singularities. An axis-agnostic, sign-less approach is employed that makes extended use of the lazy constraint building framework. Within this framework, our approach may generate insufficiently strict constraints at first, but converges towards sufficient constraints through iteration, effectively solving the IQP (eq. (7.1)) for guidance not only on what to separate but also on how to separate.



(a) (b) (c)

Figure 10.1: (a) A T-mesh (black) on a surface domain with a seamless parametrization visualized as a texture, with singularities (orange) and cuts (yellow). An arc path (green) is indicated, crossing two singularities. (b) In parameter space, the total angle around singularities differs from 360°; this affects the perceived turning angle of paths passing the singularity infinitesimally on their left or on their right. (c) Hence, relative parametric direction labels for path segments are ambiguous, as they depend on interpretation how singularities are passed.

10.2 Structure Preservation with Flexible Singularities

In the following, we present a method to determine quantization constraints that together prevent all types of structure preservation failure listed above. The procedure, outlined in algorithm 10.1, follows the iterative approach of algorithm 7.1. Consistency and non-negativity constraints ($\mathcal{A}$ and $\mathcal{B}$, see section 7.2) as well as some of the structural constraints (C_0) can be set up *a priori*, while for the remaining ones it is expedient to formulate and apply them only in a *lazy* manner. Repeatedly solving the problem with the current constraint set and adding new constraints proven to be violated by the previous solution is guaranteed to converge to a valid solution through iteration, as we show in theorem 10.1. Compared to explicitly enumerating all constraints *a priori*, it creates less overhead and admits an easier theoretical correctness analysis. In the following, we describe the novel construction of our more general and flexible constraints C.

Algorithm 10.1: LAZY QUANTIZATION

Input: T-mesh with target lengths ℓ
Output: Final quantized integer arc lengths q
$q \leftarrow \mathbf{0}$
set up $\mathcal{A}$ and $\mathcal{B}$
$C \leftarrow C_0$ // *a priori* structural constraints
repeat
  // Solve eq. (7.1) //
  Minimize $d(q, \ell)$, s.t. $\mathcal{A}q = \mathbf{0}, \mathcal{B}q \geq \mathbf{0}, Cq \geq \mathbf{1}$
  $C' \leftarrow$ LAZYSTRUCTURALCONSTRAINTS(q)
  if C' empty **then return** q
  $C \leftarrow \begin{bmatrix} C \\ C' \end{bmatrix}$

10.2.1 Preventing Feature Merge

An established way to lazily prevent the merging of features is to conduct a graph search on the T-mesh, find zero-paths between features that are witnesses for their merging, and constrain some arcs along those paths to become expanded in the next solution. With fixed singularities, this graph search can be directly based on the arc-node-graph of the T-mesh. The shape and size of features can be represented as 0D parametric points, or 1D and 2D parametric intervals. These intervals can be virtually carried along the path and checked for overlap. However, this does not work outside of regular regions, because transporting such an interval across a singularity implies ambiguous splitting or folding of the interval along that singularity (imagine e.g., an interval being transported along the path in fig. 10.1).

We therefore, instead of checking overlaps between arbitrary intervals under the quantization, consider only discrete points to simplify matters. T-mesh nodes alone however, are not sufficient

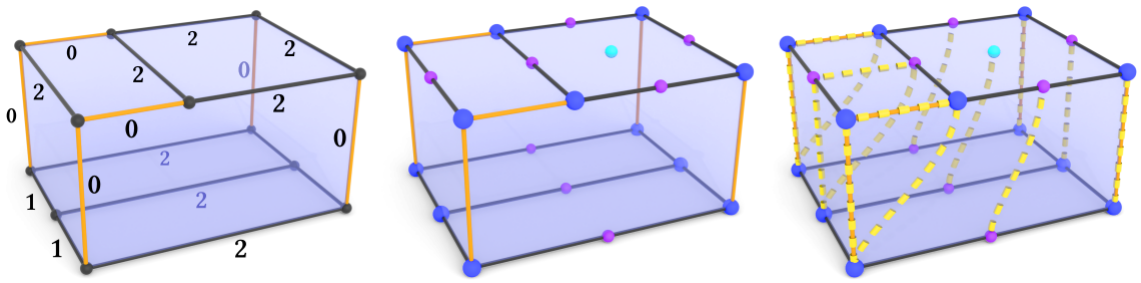
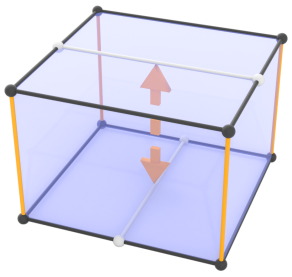


Figure 10.2: On a T-mesh with preliminary quantization integers (left), the integer grid graph is constructed by inserting integer-grid points on each T-mesh element for each implied intersection with the integer grid (center), and connecting points by links that correspond to the same integer coordinate (right).



to detect all types of feature conflicts, as demonstrated on the example on the right, where the orange arcs having 0-length implies merging of the two white feature arcs away from any node. Ideally, each T-mesh cell should be sampled exactly at those spots where the quantization implies a vertex of the later hexahedral mesh. Then, by connecting the spots on different T-mesh cells that correspond to the *same* vertex (due to a quantized distance of 0), and finding a path connecting spots on two different features implies them sharing at least one vertex in the later hex mesh, demonstrating that these features would merge. A coarser sampling would not be able to single-vertex overlaps, while other types of feature overlap (along hex edges or quads) are covered via overlap of incident vertices. A finer sampling *can* be useful to disambiguate between vertex, edge and quad overlaps, but this is not required here, as any type of feature overlap is forbidden.

We therefore suggest a graph based on the above ideas, whose nodes and connectivity are deduced from the current quantization integers, termed the *integer-grid graph*. We call the graph's nodes *integer-grid points*, which are the intersection points of any T-mesh element with the integer-grid implied by the quantization integers, and its edges *links*, locally connecting any integer-grid points corresponding to the same vertex of the implied hexahedral mesh. The construction process is illustrated in fig. 10.2 and detailed below.

10.2.1.1 Integer-Grid Graph

W.l.o.g. we assume the interior of blocks to be transition-free, with non-trivial transitions then lying on patches of the T-mesh only, and furthermore assume the integer-grid coordinate systems implied by the current quantization and the seamless coordinate system to be equally oriented. This corresponds to the notion of a local q-chart ϕ_q (cf. section 7.3.1), within which integer-grid coordinates per node can be deduced from integer arc lengths and directions.

In each block, this yields a single integer-grid point per contained node n. Contained arcs a cover integer-grid intervals of length q_a. To avoid redundancy, only integer-grid points in their interior are modelled, yielding $\max(0, q_a - 1)$ integer-grid points per arc, interpolated between the two bounding node coordinates. The integer-grid rectangle formed by patches p with quantized side lengths q_p^w and q_p^h then covers $\max(0, (q_p^w - 1)(q_p^h - 1))$ interior grid points, obtained by interpolation between the four patch corner nodes. Links are (implicitly) created within each block between any two integer-grid points sharing the same local integer-grid coordinates.

The construction process per block is summarized in fig. 10.2. Across blocks, the transitions between blocks define a correspondence between the per-block integer-grid points and thereby connect the block-wise graphs into a global one.

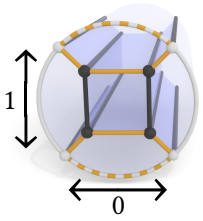
10.2.1.2 Searching the Integer-Grid Graph

By searching this graph, starting from an integer-grid point on a feature, we can quickly determine all other points on features that would collapse together under the quantization. Not all paths collapsing two different integer-grid points on features are problematic, however. Those that do not require separation specifically include:

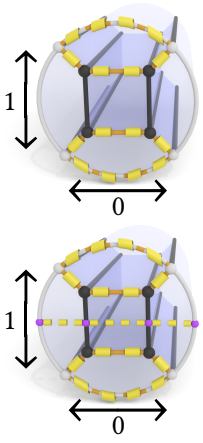
1. Paths that stay within the same feature.
2. Paths that never leave the union of start and end feature, i.e., paths between features that are adjacent via a shared boundary (or paths homotopic to those). Separating these features is infeasible, because they are already connected in the input.

Both the above can be excluded by first conquering all integer-grid points reachable by paths that never leave the starting feature or features it shares a boundary with, and marking these points as unproblematic. Then the actual search can be conducted, creating a separation constraint for any problematic integer-grid point encountered from the starting point. One full lazy separation iteration then repeats the above, starting a search from each feature integer-grid point and adding any new separation constraints determined in these searches to the current quantization problem. Note that contrary to section 7.3 self-separation of feature integer-grid points along cycles is not required here, because incontractible cycles are explicitly handled later. The self-separation of distinct integer-grid points on the same feature, however, is included.

Non-simple features By definition, feature curves are straight and feature surfaces planar with respect to the underlying parametrization. Through transitions, they can still be topologically more complex than a segment or a disk, respectively, called *non-simple* here for short. Within non-simple features, like the cyclic feature curve (white) in the inset, it is possible that two opposite non-zero regions exist (left and right arc in the inset), whose boundaries (white nodes) are implied to collapse onto each other *within* the containing feature, due to being part of the same zero-regions. These collapses, due to being executable within the feature (orange-white in the inset), are correctly considered unproblematic by the above routine.



However, in the case that the two non-zero regions have no integer-grid points in their interior (top in the inset), marking their merging boundaries as unproblematic inadvertently masks the potential collapse of their interiors onto each other. This interior collapse should be considered problematic, due to being executable only *outside* the feature. Fortunately, the solution within the framework of the integer-grid graph is simple: Before starting searches from non-simple features, virtually scale up the quantization integers by factor 2; this guarantees that an integer-grid point is present on the interior of any non-zero arc and patch (bottom in the inset). From these virtual interior integer-grid points a problematic collapse of arcs or patches onto each other is correctly detected, as their collapse path does not lie within the unproblematic zero-regions.



10.2.1.3 Search Optimization

Searches can be grouped by starting feature; then all points within the feature marked as unproblematic in a previous search starting from the same feature can be skipped (they correspond to the same hex vertex). Furthermore, by employing a Dijkstra shortest-path search, and discarding

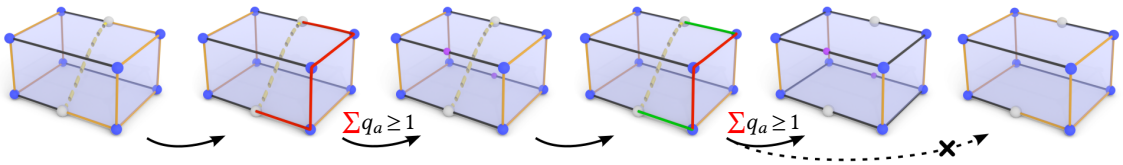


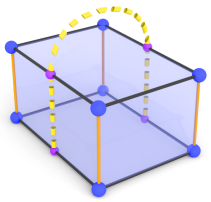
Figure 10.3: The lazy constraint building process repeatedly finds separation-worthy paths (yellow) within the integer-grid graph, translates these graph paths into arc paths in the T-mesh (red/green) and constrains the length sum over all currently zero-length arcs within the path (red) to be non-zero. Repeating this converges to a valid separation, but may sometimes cut off valid solutions (right) once the constraints become stricter.

sectors that problematic points have already been found in, as done in section 7.4, we can limit separation to nearest neighbors, significantly reducing the runtime per iteration while increasing the iteration count only slightly.

The integer-grid graph has a size that scales with the output hex mesh size. We can reduce this to scale only with the T-mesh size via the following *bundling* formalism. Recall that arcs are axis-aligned intervals and patches are rectangles under the quantization. The integer-grid points in their interior are exactly those covered by the (closed) arc-interval or patch-rectangle shrunk from all sides by 1. Thus, integer-grid points can be bundled per containing arc or patch into such 1D or 2D intervals (represented via only two extremal integer-grid points). Links can likewise be bundled: The bundle of links between two distinct point bundles corresponds to their geometric intersection. Hence, the integer-grid graph can be traversed via intersection queries between intervals, without explicitly constructing individual points and links.

10.2.1.4 Constraint Formulation

Once two points on two different features and a separation-worthy path in between, made of links in the integer-grid graph, have been identified, a constraint must be formulated based on T-mesh arcs rather than graph links. First, to convert the link path into a sequence of T-arcs, we employ another Dijkstra algorithm. For this we determine the T-nodes closest to the start and end points of the link-path based on seamless coordinates. Then the parametrically shortest arc-path between start and end nodes is computed by a Dijkstra search, confined to arcs of the blocks traversed by the link path. This would fail in the rarely occurring case that a link-path crosses the same block more than once (see the inset). In that case we instead apply the above procedure to each link of the path individually, eventually stitching these together to build the whole path. By doing so we ensure that the arc path is always homotopic to the link path.



We then constrain the sum over the quantized lengths of all 0-arcs (arcs with currently $q_a = 0$) in the path to be ≥ 1. The idea behind this sum-based way of formulating the constraint is to effectively let the solver choose which arcs are least costly to "inflate" in terms of the quantization (and even freely update this in subsequent iterations), instead of us making a pick. A key difference in this constraint formulation to that from section 7.4 is that, due to the direction ambiguity in our flexible-singularity setting as illustrated in fig. 10.1, we use a completely direction agnostic approach: The relative parametric direction of arcs in the paths does not play a role in the above constraint formulation. This is in contrast to the *directed signed* constraints used in previous work, relying on the notion of *opposite* and *transversal* orientation within the path.

As a consequence of this direction agnostic formulation, such a constraint does not always separate the two features right away, see fig. 10.3 for an example. However, subsequent iterations

of lazy constraint addition is guaranteed to take care of this eventually as detailed below.

Proposition 10.1 (Termination and feasibility). *The lazy, iterative addition of constraints based on the violation detection described above always converges in a finite number of steps to a solution, in which all features are separated.*

Proof We call a feasible solution, in which all separable features are separated a *separating* solution. A solution with no zero-arcs—or phrased differently, a solution in which no arc subset contains only zero-arcs—trivially has no zero-paths and must be separating. Otherwise, there would be a problematic path in the integer-grid graph between two non-separated features, and a homotopic path on the arc-node graph that contains at least one zero-arc, which would be a contradiction.

Each sum constraint we add exactly constrains a subset of all arcs of the T-mesh to not contain only zero-arcs. Not only this, but it constrains a subset for which so such constraint exists yet (as it is formulated exactly over the zero-arcs of a path in a previous solution). Because

- the number of unique arc subsets is finite,
- a new unique subset is added to the constraint set *iff* the previous solution was non-separating, and
- constraining all unique subsets is guaranteed to yield a trivially separating solution,

the iterative process is bound to converge to a feasible separating solution.

10.2.2 Preventing Feature Collapse

Preventing the collapse of features to lower dimensionality is applicable to 1D feature curves, represented in the T-mesh by a chain of arcs, and 2D feature surfaces, represented by a collection of neighboring patches. Preserving feature curves is easy using *a priori* constraints: we simply demand for each feature curve that the sum of quantized arc lengths along the feature curve is positive.

Preventing feature surface collapse is not as straightforward, because, e.g., demanding a positive area in a direct way would require non-linear constraints. However, as feature surfaces are bounded by feature curves by definition, and these are already constrained to length ≥ 1 each, many feature surfaces do not require dedicated attention. In particular, for feature surfaces with boundary, a non-zero parametric area of the feature surface is implied because the cycle of boundary feature curves is preserved and also prevented from degenerating to a line. For genus 0 surfaces with boundary this suffices for full topology preservation.

For closed surfaces and genus > 0 surfaces with boundary, additional care is needed for exact structure preservation. Such cases are covered in the following.

10.2.3 Preventing Topology Change

Topological Invariants A quantized length of 0 assigned to an arc or arc path effectively means a contraction of its two end points onto one another in parameter space. If this were to happen to an (arc) loop that is topologically *incontractible* (in $\mathcal{T}$ or in $\partial\mathcal{T}$), a topological mismatch between object and parametrization would be implied, as object and T-mesh (after collapsing 0-elements) were no longer homeomorphic. Loops incontractible in $\mathcal{T}$ are captured by the fundamental group $\pi_1(\mathcal{T})$ of our domain $\mathcal{T}$. Equivalently, a total quantized area of zero for topological spheres formed by patches means a compression of this sphere to a curve or point. Topologically incompressible spheres of $\mathcal{T}$, captured by the second homotopy group $\pi_2(\mathcal{T})$ must

therefore also be considered in the quantization constraints. For 3-manifolds embedded in $\mathbb{R}$ higher homotopy groups are trivial and need not be considered.

Considering the homotopy groups of feature curves and surfaces, we observe that a feature curve has at most a single unique loop, namely the feature (loop) itself, which is preserved by the constraints mentioned in section 10.2.2. Feature surfaces can have different loop classes but at most a single incompressible sphere; in the latter case this sphere *is* exactly the feature. In the following, we discuss how the following invariants can be preserved via constraints:

(i) The first homotopy group of the domain, $\pi_1(\mathcal{T})$.

(ii) The first homotopy group of a feature surface S, $\pi_1(\mathcal{T}|_S)$.

(iii) The second homotopy group of the domain, $\pi_2(\mathcal{T})$

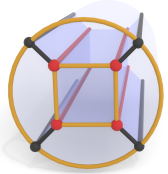
(iv) The special case of a sphere-topology feature surface.

The concrete constraint formulation is discussed afterwards.

Incontractible loops of $\mathcal{T}$ The first set of loops we consider are those from the fundamental group $\pi_1(\mathcal{T})$ of our domain $\mathcal{T}$; these are all classes of loops that cannot be contracted to a point within $\mathcal{T}$. In section 10.2.3.1 we describe how to form constraints that preserve this fundamental group by preventing the quantization from collapsing a selected covering the *generator loops* of $\pi_1(\mathcal{T})$ (cf. section 8.2.1.2). In the following, we assume that all loops of $\pi_1(\mathcal{T})$ are prevented from collapsing and assess how this simplifies preservation of other invariants.

Incontractible loops of $\mathcal{T}|_S$ The next aspect to consider are loops that are incontractible within the restriction of $\mathcal{T}$ to a feature surface S. Recall, that feature surfaces are segmented into *feature regions*, within which the parametric alignment axis is constant. Because the boundaries of such feature regions are mutually separated (preventing zero-loops across feature region boundaries), we can restrict our considerations to feature regions S, which are parametric isoplanes.

Consider a loop $l \in \mathcal{T}|_S$ that is incontractible within the restricted T-mesh, $\mathcal{T}|_S$. If that same loop is also incontractible within the whole T-mesh $\mathcal{T}$, then it is already covered by the preservation of $\pi_1(\mathcal{T})$. Otherwise, the loop is contractible within $\mathcal{T}$, and contracting it to a point would sweep a disk-like surface within $\mathcal{T}$. The boundary of this disk is parametrization-aligned due to lying in a feature region. Then, by the Gauss-Bonnet (or the Poincaré-Hopf theorem for an equivalent cross field [Ray et al. 2008]), the seamless map's restriction to the swept disk must be pierced by singularities of total index -4, manifested as point singularities within the disk. In practically relevant parametrizations, singularities of index -2 (or lower) typically do not occur; these would induce hex mesh edges of valence ≤ 2 and non-convex hexahedra (with inner angles $\geq 180°$) that are at least unfavorable, if not completely unsuitable for downstream applications. Thus, we generally assume inputs to contain no singularities with index below -2 (in fact even singularities with index -2 do not occur in any dataset processed in the scope of this thesis). Under this restriction there are at least two distinct singularities within the disk.



The inset shows such a disk, bounded by a non-contractible loop on a feature surface, and in this case containing four singularities of index -1 in red. If the arcs highlighted in orange are zero-arcs, the (incontractible) loop of the feature surface collapses. Such a collapse can happen only if negative-index singularities merge into a singularity of even lower index; preventing this is advisable anyways, as argued above. Note that merging of negative-index singularities is not generally problematic, as this could be compensated by additional positive-index singularities

merging into the same cluster of singularities. Only if the cluster of merged singularities has an index sum that is lower than the minimum index of its constituents, the cluster must be split. In section 10.2.3.2 we describe how to formulate constraints that keep the index sum of merged singularities within fixed bounds. This avoids unfavorable singularities in the output and additionally preserves all incontractible loops of feature surfaces, as discussed above.

Incompressible spheres of $\pi_2(\mathcal{T})$ The next aspect to consider are cases where the second homotopy group $\pi_2(\mathcal{T})$ of the whole domain $\mathcal{T}$ is non-trivial, i.e., when there exist incompressible spheres in $\mathcal{T}$. Such incompressible spheres indicate the presence of volumetric holes (voids) in the interior of $\mathcal{T}$. Each such void is surrounded by a connected component $S \in \partial\mathcal{T}$ of the domain boundary, which by our definition is a feature surface. If an incompressible sphere contains more than a single void, then the separation of the void boundaries also preserve the sphere. Otherwise, if it contains only a single void, then we need only consider the boundary feature S around this void. If S has genus > 0 the constraints discussed above already preserve its topology, hence prevent its collapse, and that of any sphere surrounding it. This leaves only the case of S being spherical, which is covered below.

Spherical feature surfaces S If S consists of more than a single feature region, separation of the inbetween feature links and nodes prevents the collapse of S. This leaves only the single special case of S being a single spherical feature region, aligned with a constant axis of the parametrization. Within the restriction $\mathcal{T}|_S$ the input seamless map is the 2D surface parametrization of a sphere. The Gauss-Bonnet theorem demands a sum of singularity indices equal to -8 for such a map. The point-like singularities implied by this condition extend to singular curves within $\mathcal{T}$, for which the singularity cluster index constraints (section 10.2.3.2) ensure that there are enough separate ones remaining, that together span a non-zero area for the sphere. Hence, by preventing the collapse of incontractible loops in the domain $\mathcal{T}$ and maintaining a fixed lower bound for merged singularities, the topology of both the entire domain as well as individual feature surfaces is preserved. We discuss how to achieve these two goals in the following.

10.2.3.1 Preservation of $\pi_1(\mathcal{T})$

Quantizations implying the contraction of loops that are incontractible in $\mathcal{T}$ need to be prevented. The idea is to detect when a connected component in the integer-grid graph, i.e. a zero-region, is not simply connected. This indicates zero-loops wrapping around topological features (violations of incontractibility) or around certain clusters of negative-index singularities (violations of index bounds). Explicit distinction is not necessary as both must be prevented; the latter is also (more generally) handled in section 10.2.3.2. Other types of regions cannot be wrapped by a zero-region as consistency constraints would transitively imply them being part of the zero-region.

A non-simply connected zero-region in the integer-grid graph can be detected by the following procedure, akin to a tree-cotree decomposition [Eppstein 2003]. First, compute a spanning tree of the zero-region. Let the co-tree contain all untraversed links of the region. Each of these links, when inserted into the tree, would form a loop – contractible (within the zero-region) or incontractible. To exclude the contractible ones, we remove links, one by one, from the co-tree (inserting them into the former tree) if they close any loop that is entirely contained within one T-mesh block – which trivially implies its contractibility. This is iterated so as to, transitively, also exclude links closing larger contractible loops. Afterwards, for each remaining link in the co-tree, for the root-based loop it closes, a constraint is formulated as described in section 10.2.1.4, eventually enforcing the inflation of this loop in the quantization.

This does not need to be done everywhere. Instead, consider a *cut surface* that virtually cuts $\mathcal{T}$ into a topological ball. Any incontractible loop must pass through this cut surface, so searching only from integer-grid points on this cut surface is sufficient. To compute a discrete cut surface $C \subset P$ we initialize C with all patches P, perform a breadth-first search on the dual of the T-mesh $\mathcal{T}'$, removing crossed patches from C. Then "dead-end" patches, containing any arc incident to no other *cut* patch, are removed from C iteratively, and finally boundary patches are removed.

10.2.3.2 Singularity Index Bounding

As we do not treat the input singularity structure as fixed, we need to take care that only permissible modifications occur. When singular curves merge (due to not being separated by the quantization) the resulting curve has an index that is the sum of the original indices. We wish to exclude mergings that lead to combined indices out of a range $[I_{\text{low}}, I_{\text{high}}]$ of acceptable indices. Most importantly, we set $I_{\text{low}} = -1$ as singularities of lower index imply hex mesh edges of valence ≤ 2 and non-convex hexahedra. I_{high} is an optional, adjustable upper index bound.

We employ a variation of the integer-grid graph (section 10.2.1.1) for detection and separation of clusters of singularities merged by the current quantization that have out-of-bounds indices. As opposed to the previous considerations on feature separation, we do not care for vertex-based coincidences, as two singular curves joining merely at a common node are not relevant here. Instead, we care about singular curves overlapping along a 1D segment. The shortest possible such segment is one edge of the implied hexahedral mesh. So the integer-grid graph must be adapted to be based on implied edges, rather than vertices. As discussed before, this can be achieved by virtually doubling the quantized arc lengths. Then, within each arc segment of length one, there is an additional integer-grid point in the center of that segment.

On the double-resolution integer-grid graph, we start a search rooted at an edge-representative integer-grid point on a singularity. The resulting search tree, made from all integer-grid graph links on shortest paths starting from the root, is pruned to a spanning tree of all the reached singular integer-grid points (which all correspond to edge-overlaps in the implied hex mesh). We call the set of these singular integer-grid points a *singular cluster*, and each chain of integer-grid graph links connecting two of such points within the spanning tree a *cluster chain*. Any singularities within the singular cluster would merge under the quantization, leaving a single singular (or regular) edge with an index equal to the cluster's index sum. That index sum I is determined and compared with the index bounds $[I_{\text{low}}, I_{\text{high}}]$. If I is within bounds, we leave it as is. Otherwise, the index distance ΔI to the closest bound is computed. For each cluster chain, we compute the index sums I_1 and I_2 of the two sub-clusters it connects. Splitting any chain for which $\max(\Delta I_1, \Delta I_2) < \Delta I$ brings the worst index closer to within the bounds; repetition results in sub-clusters with only inside-bounds indices.

Here again, we leave the choice of which link to split to the solver minimizing the outer objective, instead of making a narrowing, heuristic choice. To do this, for each chain where $\max(\Delta I_1, \Delta I_2) < \Delta I$, we transform its link path into an arc path as discussed in section 10.2.1.4, and then simply formulate a constraint that demands the sum over all lengths of 0-arcs in those paths to be ≥ 1. Repeating this will eventually dissolve or recombine all out-of-bounds singularity clusters, leaving only valid ones. Note that within the above, we have assumed that the input singularities meet the index bounds. In case the input contains out-of-bounds singularities, they either improve through objective-driven merging, or remain unchanged, but will never worsen.

10.3 Flexible Quantization Objective

The goal of the quantization's objective function is to minimize the implied deviation of the output integer-grid map from the input seamless map. In previous chapters we used an objective function that punishes the deviation of each individual arc's length from its target. Specifically, eq. (7.1a) becomes

$$d_{\text{arc}}(q, \ell) = \sum_{a_i \in A} (q_i - \ell_i)^2 = (q - \ell)^\mathsf{T} \mathcal{I} (q - \ell) = ||q - \ell||_{\mathcal{I}}^2 , \tag{10.1}$$

where $\ell \in \mathbb{R}^{|A|}$ and $q \in \mathbb{Z}^{|A|}$ are vectors containing the continuous seamless length and quantized integer length per arc, and $\mathcal{I}$ is the identity matrix. However, the lengths of individual arcs should not actually matter to us—they do not carry over to the final IGM or hexahedral mesh, as discussed in chapter 8. They may be relevant for a block-wise initialization of the IGM, but become irrelevant once a global smoothing optimization is applied, as illustrated in fig. 9.17. Only the *accumulated* integer spacings between and along features matter, as these features are fundamentally fixed in object space.

For the goal of maintaining feature sizing and spacing, the arc-based objective can even be strongly detrimental. Imagine two features separated by a straight chain of N arcs, each with target length $\ell_i = 0.4$. Under the above objective, assigning an integer length of 0 to each, and thus a total length of 0 to the chain, is considered optimal. Instead, the objective should only consider the relative spacings between features, as only these are invariant in the later steps of the meshing pipeline, and encourage the chain of arcs to have a total length of as close to $0.4N$ as possible in this example. In particular, the result should not be strongly influenced by the density of arcs in the domain, which is related to the density of singularities.

To generalize this intuition, we define in the following a novel *feature-based* rather than *arc-based* (and thereby indirectly also singularity-based) objective, focused on paths between features; it is independent of the density of the T-mesh arc sampling induced by the singularity structure and thus serves as a high-level decision maker for the overall goal of adaptively simplifying the singular structure wherever beneficial. To maintain the benefits and simplicity of a quadratic objective, our objective is of similar form,

$$d_{\text{feature}}(q, \ell) = (q - \ell)^\mathsf{T} \mathcal{F} (q - \ell) = ||q - \ell||_{\mathcal{F}}^2 . \tag{10.2}$$

The matrix $\mathcal{F}$ is responsible for accumulating individual arcs into feature-spanning and feature-connecting paths, as well as weighting these appropriately. As a conceptual basis, we first introduce a general measure of feature distortion between integer-grid map and seamless map and then show how that measure is efficiently approximated on top of our T-mesh.

10.3.1 Feature Distortion Measure

When comparing seamless and quantized integer-grid map, represented as target and quantized T-mesh arc lengths, our overall goal is to measure and minimize the distortion of size and relative position of features. Relative positions of features can be assessed based on paths between these. Due to transitivity, it is sufficient to consider paths between neighboring features instead of paths between all pairs. The notion of neighborhood should be as tight as possible to avoid redundancy, but sufficiently broad to not leave gaps in the resulting path network. As a natural neighborhood notion, we suggest the use of Voronoi adjacency, i.e. two features are considered neighboring if their cells in a volumetric Voronoi diagram (in the metric of the seamless map) are adjacent. Based on the resulting path network we can measure feature distortion, and formulate a volume

integral of this measure to make it agnostic to network density. Both path length and direction are important to preserve, because spacing per axis matters. Concretely, let E be the set of paths, $\vec{e}_\ell$ and $\vec{e}_q$ the vector from start to end of $e \in E$ in (a local chart of) the seamless map and quantized map, respectively, and $V_\ell(e)$ some part of the volume (in the seamless map metric) exclusively assigned to it, the suggested relative distortion measure to minimize then is

$$d_{\text{feature}}(q, \ell) = \sum_{e \in E} V_\ell(e) \frac{\left|\left|\vec{e}_q - \vec{e}_\ell\right|\right|^2}{\left|\left|\vec{e}_\ell\right|\right|^2}. \tag{10.3}$$

In the following we clarify how to express $V_\ell(e)$, $\vec{e}_\ell$ and $\vec{e}_q$ in terms of arc-based targets ℓ and quantization variables q.

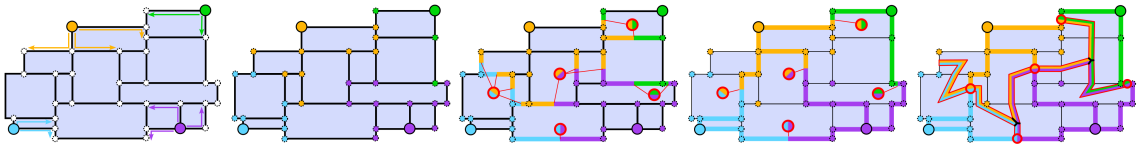
We remark that the path network derived from Voronoi adjacency is akin to a conforming Delaunay tetrahedrization of the features—with the caveat that the latter is not entirely well-defined in the metric of the seamless map due to the singularities. Interestingly, the utility of a Delaunay tessellation for quantization was pointed out before, for the 2D case [Coudert-Osmont et al. 2024]. Importantly though, in that work it was used differently, as an alternative to the T-mesh, limiting it to the 2D surface setting. Here we instead employ it as a guiding principle on top of the T-mesh, as detailed next.

10.3.2 Approximation on the T-Mesh

We construct a set E of paths, guided by the above concept, as chains of arcs by computing approximate Voronoi cells directly on the basis of the discrete T-mesh structure.

Voronoi diagram As seeds for the Voronoi cells we use feature nodes. As feature curves are bounded by such nodes, and feature surfaces are bounded by feature curves bounded by such nodes, in this way the resulting paths conveniently capture both inter-feature spacing as well as intra-feature sizing. To properly include node-less and node-poor features in this, on feature curves with ≤ 2 feature nodes, i.e., cyclic ones, and on feature surfaces with ≤ 4 feature nodes, we greedily add maximally distant pseudo feature nodes.

From these (pseudo) feature nodes, concurrent Dijkstra shortest path searches on the node-arc-graph of the T-mesh are started, using ℓ as arc weights. T-mesh nodes are colored according to the seed point they were first reached from, see fig. 10.4. Voronoi cells are formed by (dual cells of) nodes of the same color. Each arc connecting two nodes of different color is marked as an interface arc. Interface arcs are then grouped into *interfaces*, connected components (where arcs are considered connected if adjacent to a common patch) with constant color pairing. Each such interface corresponds to a face of the discrete volumetric Voronoi diagram, shared by two

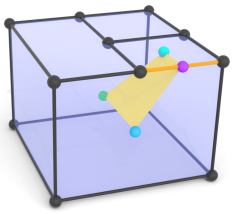


(a) (b) (c) (d) (e)

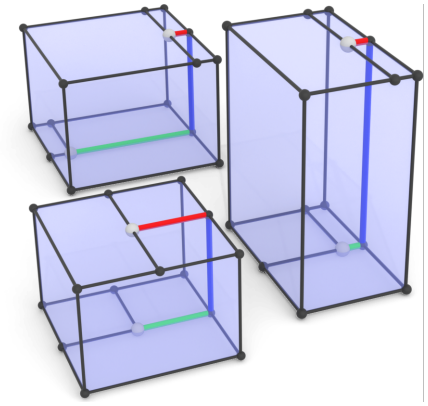
Figure 10.4: The T-mesh based approximation of a feature-based Voronoi diagram of the parameter space and resulting Delaunay edges between features, illustrated on a 2D example. (a) Starting from feature nodes, Voronoi regions are grown via arc paths. (b) Regions are formed by all nodes of the same coloring. (c) Interfaces between regions are arcs with different color pairings at their end nodes. (d) The shortest path through each interface arc group is determined. (e) The interface area is estimated based on the (barycentric) dual of interface arcs.

cells. Lastly, to form the set of arc-paths E, the shortest arc-path through each such interface connecting its two seeds is determined.

Volume integral The seamless map volume $V_\ell(e)$ pertaining to each such path $e \in E$ can be conveniently estimated as $V_\ell(e) = \frac{1}{3} A_\ell(e)\, ||\vec{e}_\ell||$, where $A_\ell(e)$ is the area of the interface traversed by e, i.e., the sum of areas of dual faces of the interface's arcs in the T-mesh's barycentric dual. This is computed per interface arc a (orange) as follows: For each incident block b_a compute the parametric dual area attributed to a, by triangulation of the quad spanned by the midpoint of a (purple), the midpoints of the two patches incident on both a and b_a (cyan), and the midpoint of b_a (green). The idea behind this is an imagined partitioning of a Voronoi cell into (polygonal) cones, apex at the seed, with the interfaces as bases, attributing a cone's volume to the corresponding interface's path.



Path vectors The vectors $\vec{e}_\ell$ and $\vec{e}_q$ of an arc path are not uniquely defined, in particular when crossing singularities. The 3D parametric difference vector between start and end node is ambiguous due to the lack of a global regular coordinate system, as in fig. 10.1. We could circumvent this by just using the sum of the path's arcs' lengths, ignoring relative orientation. But this would approximate the vector length only crudely and would ignore directional information entirely. The inset demonstrates three different quantizations of the same T-mesh block. Despite the two white feature nodes being in very different spatial relation, the sum of arc lengths along the red-blue-green arc path is the same. To actually discern these, the information that the blue arc lies along a different axis than red and green, and the information that red and green arcs are in opposite orientation along the path, is important.



We heuristically disambiguate the relative directional ambiguity when an arc-path from E crosses a singularity, by assuming it infinitesimally passes by the singularity, and of the multiple ways there are to pass a singularity (two in the case of singular curves, more in the case of singular nodes) picking a shortest one, i.e. smallest turning angles. For this purpose, lengths and angles are measured with respect to the input seamless map, represented by arc lengths ℓ and (per-block) arc orientations. The rationale behind this is that the arc-paths in E are (discrete approximations of) parametrically shortest paths. In this way axis direction and orientation labels for all arcs along a path, relative to the path's first arc, are determined. We express the resulting axis and orientation information per arc a_i in an arc-path e by defining for each triplet (e, a_i, dir), with $dir \in \{\vec{u}, \vec{v}, \vec{w}\}$, a ternary sign:

$$s_{e,i}^{dir} = \begin{cases} 1 & \text{if within } e \;\; a_i \text{ is traversed along } dir \text{ in pos. direction} \\ -1 & \text{"} \qquad\qquad\;\; a_i \text{ is traversed along } dir \text{ in neg. direction} \\ 0 & \text{"} \qquad\qquad\;\; a_i \text{ is not traversed along } dir \end{cases}$$

Using this we can now express the edge vector difference in terms of arc variables q_i:

$$\left\|\vec{e}_q - \vec{e}_\ell\right\|^2 = \sum_{dir \in \{\vec{u},\vec{v},\vec{w}\}} \left(\sum_{a_i \in A} s_{e,i}^{dir} (q_i - \ell_i) \right)^2 .$$

Final objective Using the above, (10.3) is expressed in arc variables:

$$\begin{aligned} d_{\text{feature}}(q,\ell) &= \sum_{e\in E} w_e \sum_{dir} \left(\sum_i s_{e,i}^{dir} (q_i - \ell_i) \right)^2 , \\ w_e &= \frac{V_\ell(e)}{||\vec{e}_\ell||^2} = \frac{A_\ell(e)}{3\,||\vec{e}_\ell||} = \frac{A_\ell(e)}{3\sqrt{\sum_{dir} \left(\sum_i s_{e,i}^{dir}\, \ell_i \right)^2}} . \end{aligned} \tag{10.5}$$

It can be rearranged into matrix form

$$d_{\text{feature}}(q,\ell) = (q-\ell)^\mathsf{T} \mathcal{S}^\mathsf{T} \mathcal{W} \mathcal{S} (q-\ell) = (q-\ell)^\mathsf{T} \mathcal{F} (q-\ell) , \tag{10.6}$$

where the matrices $\mathcal{S}$ and $\mathcal{W}$ are effectively responsible for accumulating arcs into arc paths and weighting the paths, respectively:

$$\mathcal{S} = \begin{pmatrix} s_{1,1}^{\vec{u}} & s_{1,2}^{\vec{u}} & \cdots & s_{1,|A|}^{\vec{u}} \\ s_{1,1}^{\vec{v}} & s_{1,2}^{\vec{v}} & \cdots & s_{1,|A|}^{\vec{v}} \\ s_{1,1}^{\vec{w}} & s_{1,2}^{\vec{w}} & \cdots & s_{1,|A|}^{\vec{w}} \\ s_{2,1}^{\vec{u}} & s_{2,2}^{\vec{u}} & \cdots & s_{2,|A|}^{\vec{u}} \\ \vdots & \vdots & \ddots & \vdots \\ s_{|E|,1}^{\vec{w}} & s_{|E|,2}^{\vec{w}} & \cdots & s_{|E|,|A|}^{\vec{w}} \end{pmatrix} ,$$

$$\mathcal{W} = \operatorname{diag}\left(w_1, w_1, w_1, w_2, w_2, w_2, \ldots, w_{|E|}, w_{|E|}, w_{|E|} \right) .$$

Note that each arc path in E often extends along only one or two axes, so $\mathcal{S}$ may contain many zero-rows; these rows and the corresponding rows and columns in $\mathcal{W}$ can be dropped. Furthermore, matrix $\mathcal{F}$ commonly is not full-rank as not all arcs are covered by E. It is therefore sensible to use the arc-based objective (10.1) as a regularizer,

$$\begin{aligned} d_{\text{total}}(q,\ell) &= d_{\text{feature}}(q,\ell) + \lambda\, d_{\text{arc}}(q,\ell) \\ &= ||q-\ell||^2_{\mathcal{F}+\lambda\mathcal{I}} , \end{aligned} \tag{10.8}$$

with λ chosen adequately small (e.g. $10^{-3} V_\ell$) so that the second term serves only as a *tiebreaker*. This avoids numerical problems in common optimization schemes and favors solutions with lower arc stretch from among those with equal distortion.

10.4 Results

We have implemented the method discussed in this chapter as optional components for our open source libraries QGP3D and C4HEX mentioned in sections 7.5.1 and 9.5.

10.4.1 Experiments

We apply our method to those datasets of seamless parametrizations employed in the previous chapters and compare our results to the ones obtained in these. Out of the 197 total models, 103 come with feature markers and 60 additional ones contain sharp boundary singularities that we treat as feature curves, yielding 163 models with multiple feature types and 34 models without features (except for their boundary, implicitly considered a feature surface). To cover a wide range of scenarios, for each of these models we target seven different output resolutions, by creating

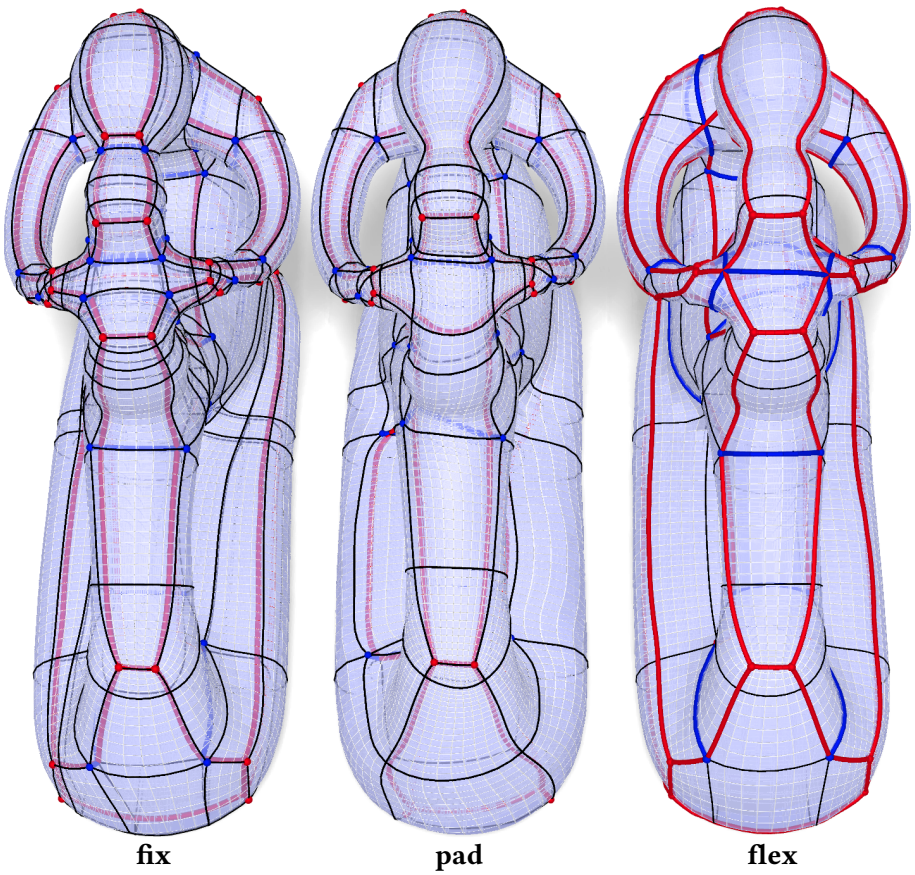


Figure 10.5: Base complex induced by three different quantization modes for handling singularities: Fixed (left), flexible singularities except that inner singularities must not merge into the boundary (center), and completely flexible (right). Valence 3 and 5 singularities in red and blue, respectively. The center option is interesting for applications that may suffer from singular curves on the boundary.

seven instances per model with the seamless map isotropically scaled to a parametric volume of $V_\ell \in \{10^0, 10^1, ..., 10^6\}$. Note that this volume V_ℓ indicates the target number of hexahedra to roughly be expected in the output.

For each of the 1372 instances, in order to thoroughly analyze our method's ingredients, we compute quantizations in each of the following five ways:

$q_{\mathbf{arc,fix}}$ Minimize the arc-based objective with fixed singularities (as in previous chapters).

$q_{\mathbf{feat,flex}}$ Minimize the feature-based objective with flexible singularities (novel contribution).

$q_{\mathbf{feat,pad}}$ Minimize the feature-based objective with flexible singularities, but maintain padding of inner singularities (novel contribution with singularity padding).

$q_{\mathbf{feat,fix}}$ Minimize the feature-based objective with fixed singularities (ablation scenario I).

$q_{\mathbf{arc,flex}}$ Minimize the arc-based objective with flexible singularities (ablation scenario II).

Figure 10.5 demonstrates the effect of padding interior singularities; singularities may still be recombined and fully collapsed, but remain separated from the boundary if they are separate in the input. As index bounds for singularities we use $[-\frac{1}{4}, \frac{2}{4}]$. From the quantizations hexahedral meshes are generated using the pipeline detailed below.

10.4.2 Complete Pipeline

We employ the pipeline summarized in section 9.7, but replace the fixed-singularity T-mesh quantization in step **IVc** by one of the variants listed above; all variants follow the algorithm algorithm 10.1 but use different structure-preservation constraints and a different objective function, as described in sections 10.2 and 10.3, respectively. As a solver for the central integer quadratic program, we use the same general-purpose solver as in section 7.6 [Gurobi Optimization, LLC 2025], configured with a time limit of 10 minutes per solve, returning the best available solution at that cut-off. For all instances, a feasible solution was found within this time limit.

T-mesh collapsing Any T-mesh elements quantized to a size of 0 are collapsed using the collapse operators from chapter 9. We adjust the original implementation so that it (optionally) does not assume singularities to be unmovable but also executes the collapsing and merging of singularities as prescribed by the quantization.

Block structuring Results of particularly coarse quantizations are rather suited as hex layouts than directly as hex meshes. To enable their visualization, we can refine them, yielding block-structured hex meshes with the coarse quantization serving as block structure (see, e.g., fig. 9.2(d)). To this end, after collapsing all zero-elements, we compute a second, now purely positive quantization (without zeros, thereby preserving the determined coarse structure), targeting a set finer resolution. We demonstrate such hex layouts and corresponding block-structured meshes in fig. 10.8.

Integer-grid reparametrization With the quantization values determined, an injective integer-grid map is built using per-block reparametrization as in section 9.6.2.1, and the resulting map globally smoothed by minimizing the symmetric Dirichlet energy [Smith and Schaefer 2015] while keeping the integer-alignment of singularities and features fixed via constraints derived from the T-mesh as in chapter 8. Hexahedral meshes are extracted from these maps [Kohler et al. 2025], and not processed any further.

10.4.3 Quantization Quality

To assess the performance of the flexible-singularity quantization in comparison to the other variants we can not simply compare the quality of the generated integer-grid maps, because a given method may achieve lower distortion by simply deviating arbitrarily from the target resolution. Instead we compare how well the quantization matches our initial target, based on two measures:

Global resolution accuracy Both the target number of hexahedra and actual number of hexahedra are encoded in the volumes V_ℓ and V_q of the input seamless map and output integer grid map, respectively. The higher flexibility and more global scope of our method should result in higher accuracy, i.e., generally smaller gaps between the two resolutions.

Feature distortion To also take into account local scale and alignment distortions, but only where relevant, we measure the feature distortion via eq. (10.5). Our novel quantization should perform better under this metric, as lowering it is its main objective. However, it still remains to be seen whether this shift in focus actually leads to significantly different results compared to minimizing an arc-based objective, or whether the two objectives mostly align in practice. Also, this measure is useful to demonstrate specifically the benefit of flexible singularities over fixed ones.

For any two given quantizations q and q' on the same seamless input map with targets ℓ, the measures $\frac{|V_q - V_\ell| - |V_{q'} - V_\ell|}{V_\ell}$ and $\frac{d_{\text{feature}}(q',\ell)}{d_{\text{feature}}(q,\ell)}$ indicate how much more accurately q' achieves the target resolution and how much lower its feature distortion is, respectively.

The former fixed-singularity methods have a different (higher) theoretical lower limit of resolution than a flexible-singularity method. In the accuracy comparisons we wish to not punish these for their mere inability of reaching coarseness. Hence, in cases where V_ℓ is below the lowest achieved quantized volume $V_q^{\min}$ of the method on a given model (cf. table 10.1), this lowest-achieved volume is used as the reference point for accuracy computation instead.

Improvement over previous works Over all 1372 test instances, our quantization $q_{\text{feat,flex}}$ is on average 17% closer to the target mesh resolution compared to the quantization used in recent works $q_{\text{arc,fix}}$. We note, that this percentage is relative to the target resolution. When considering only the three coarsest target resolutions, the accuracy is even 30% better, as this is where flexibility of the singular structure matters most. However, even when averaging over the finer instances with target volumes of 1000 or higher, there still is a significant difference of 7%. Even when preserving the singularity padding with $q_{\text{feat,pad}}$ the improvements are still 10%, 16% and 5% over all instances, the coarser and the finer ones, respectively. Considering the same threefold grouping of instances, the feature distortion of $q_{\text{arc,fix}}$ is on average higher by factors of 5.3, 9.0 and 2.6 than that of $q_{\text{feat,flex}}$, and still higher by factors of 3.9, 6.0 and 2.3 than that of $q_{\text{feat,pad}}$, respectively.

Timings With GUROBI, computing $q_{\text{feat,flex}}$ on all test instances took about 18 hours of total CPU time, and at most 50 minutes per instance. The vast majority, 90% of instances, each took less than 40 seconds to process. The ISP solver, being only 5% less optimal on average, $q_{\text{feat,flex}}$ needed only about 4 hours in total, no more than 15 minutes per instance and no more than 15 seconds for 90% of instances. On the same machine, our quantization $q_{\text{feat,flex}}$ was about half as fast as the quantization routine $q_{\text{arc,fix}}$ from chapter 7, which needed a total of 10 hours and 2 hours using GUROBI or ISP, respectively. This difference is to be expected, as our method employs more specific—and thus harder to construct—structure-preserving constraints than methods based on singularity separation.

10.4.4 Ablations

Flexible singularities To demonstrate the benefits of a flexible singular structure in isolation, we can compare between fixed and flexible scenarios employing the same objective function. When using our novel feature-based objective function, the lack of flexible singularities in $q_{\text{feat,fix}}$ results in output resolutions that are on average 12% further off the target than for $q_{\text{feat,flex}}$. Even though $q_{\text{feat,fix}}$ focuses on minimizing the feature distortion, it is still higher by a factor of 4.3 on average. When instead comparing scenarios using the arc-based objective, $q_{\text{arc,fix}}$ still is 15% further from the target resolution than $q_{\text{arc,flex}}$. The flexible singularities in $q_{\text{arc,flex}}$ even admit a 50% lower arc-length deviation on average.

The concrete increase in flexibility introduced by the possibility to collapse and merge singularities can be quantified by comparing the coarsest conforming block layout (also called *base complex*) available for fixed-singularity versus flexible-singularity quantizations. The number of blocks in these layouts corresponds exactly to the minimum integer-grid map volume obtainable by the respective quantization. Comparing these minimal map volumes, listed in table 10.1, quantizations with flexible singularities are able to generate conforming block layouts coarser by a factor of over 100 in some cases, and a factor of over 10 in about a fifth of cases. While the

Table 10.1: Minimum integer-grid map volume with fixed, flexible or boundary-padded singularities; equivalently: number of blocks in the coarsest possible conforming block layout.

Model	$V^{\min}_{q_{\text{fix}}}$	$V^{\min}_{q_{\text{flex}}}$	$\frac{V^{\min}_{q_{\text{fix}}}}{V^{\min}_{q_{\text{flex}}}}$	$V^{\min}_{q_{\text{pad}}}$	$\frac{V^{\min}_{q_{\text{fix}}}}{V^{\min}_{q_{\text{pad}}}}$
2016 ROCKERARM-HEX	621	3	207.00	7	88.71
2011 BUMPY-TORUS	612	4	153.00	43	14.23
2022 ROCKERARM	450	3	150.00	11	40.91
2012 ROCKERARM	476	4	119.00	13	36.62
2016 CUBESPIKES-MODEL-IN	111	1	111.00	7	15.86
2013 BUNNY-HEX-OPT	215	2	107.50	19	11.32
2013 BU-HEX-OPT	268	3	89.33	17	15.76
2022 KITTEN	176	2	88.00	12	14.67
2022 CAMILLE-HAND	84	1	84.00	6	14.00
2016 HOLLOW-EIGHT-HEX	391	6	65.17	12	32.58
2022 N09C-PYRAMID	48	1	48.00	5	9.60
2022 N10U-QTORUS-CYL	187	4	46.75	14	13.36
2022 ARMADILLO	747	18	41.50	27	27.67
2018 EXAMPLE-2	570	14	40.71	76	7.50
2022 BONE	70	2	35.00	7	10.00
2012 ELLIPSOID-A	34	1	34.00	7	4.86
2016 KITTEN-HEX	119	4	29.75	7	17.00
2022 N10C-QTORUS-CYL	94	4	23.50	14	6.71
2014 ROD	212	10	21.20	24	8.83
2014 KITTY	121	6	20.17	30	4.03
…					

minimum map volume is generally higher when singularities are flexible but padded from the boundary, it still is lower compared to that of fixed-singularity maps by factors of 10 or more in several cases.

While generating a hexahedral mesh at exactly this minimal resolution is seldom useful, block-structured meshes, characterized by a coarse block layout but a finer mesh within each block, are desirable for many applications. Such block-structured meshes are shown in fig. 10.8 for block layouts stemming from $q_{\text{feat,flex}}$ or $q_{\text{arc,fix}}$, side-by-side on the same instance. Block structures generated by our method are generally much simpler. Figure 10.6 shows another such comparison and demonstrates that the coarser block layout is indeed achieved through a significant simplification of the underlying singular structure. This simplification never comes at the expense of feature preservation, even for extreme simplifications of the singular graph, as illustrated in fig. 10.7 for an embedded feature surface.

Feature-based objective Lastly, we also investigate the impact of our feature-based objective function on quantization quality. With singularities considered fixed, $q_{\text{arc,fix}}$ misses the target resolution by 5.0% more than $q_{\text{feat,fix}}$; with flexible singularities, $q_{\text{arc,flex}}$ misses the target resolution by 2.0% more than $q_{\text{feat,flex}}$. While the overall impact of the objective function on global resolution accuracy is evidently smaller than that of a more flexible singular structure, the numbers still show some advantage of our feature-based objective in that regard.

Its main virtue, however, is that it more faithfully preserves the spacing and sizing of features when compared with the unnecessarily fine-grained arc-based objective. Indeed, for both fixed and flexible singularities, minimizing the arc-based objective yields a feature distortion that is higher by a factor of 1.6 than when minimizing our feature-based objective, showing that the global distortion can often be improved if arcs deviate further from their individual targets. Figure 10.9 shows typical cases, where the feature-based objective is able to not only match the target resolution much more closely, but also maintains a globally consistent spacing and sizing

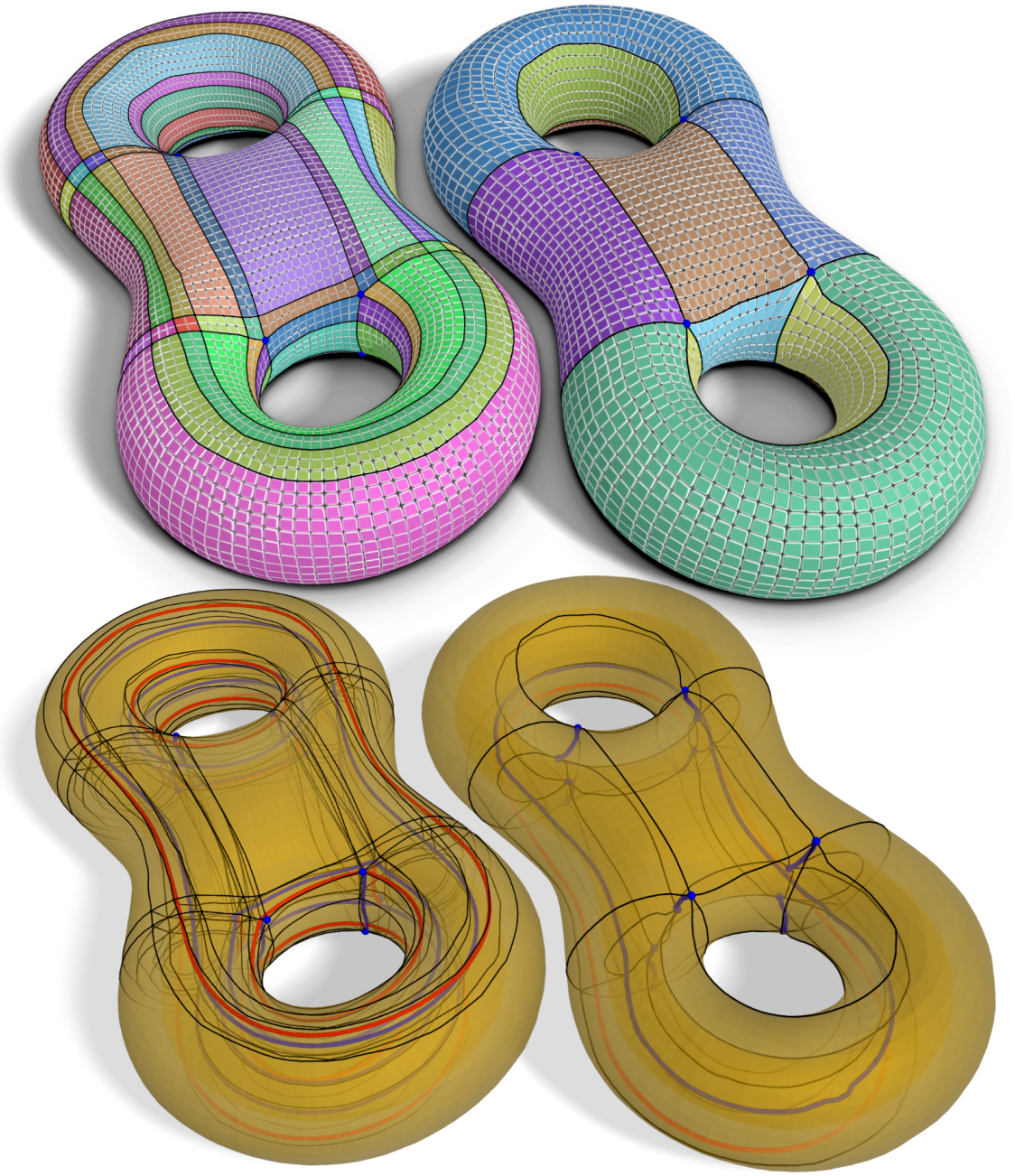


Figure 10.6: Top: side-by-side comparison of the block structure of hexahedral meshes generated by the quantization $q_{\text{arc,fix}}$ from chapter 7 (left) and our quantization $q_{\text{feat,flex}}$ (right) on the same input. Bottom: side-by-side comparison of the singularity graph embedded in the block layout, with valence 3 singularities in red and valence 5 singularities in blue. $q_{\text{arc,fix}}$ keeps the original one (left), our method simplifies it in the quantization process (right).

of features, while the the arc-based objective fails to accomplish either.

10.5 Discussion

We have introduced a quantization routine for volumetric seamless parametrizations that adaptively simplifies the underlying singular structure, depending on the target mesh resolution, while accurately preserving shape and features in the output integer-grid map. Applying this quantization in a hexahedral meshing pipeline, where former approaches fixed the singular structure in an

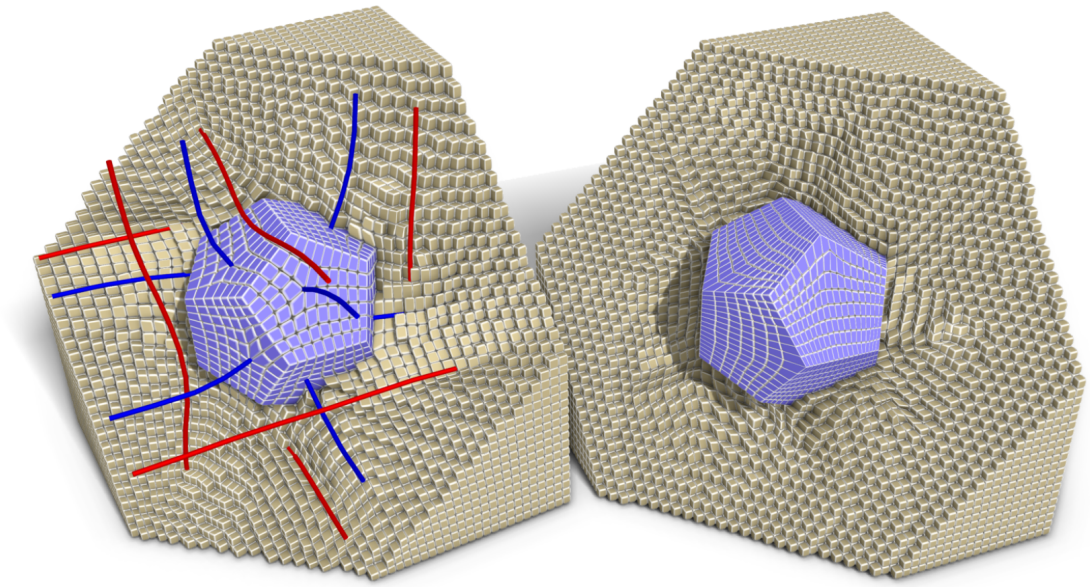


Figure 10.7: Hexahedral meshes generated from a model with embedded feature surface by our method, targetting a finer block layout (left) or the coarsest possible block layout (right). Valence 3 (red) and 5 (blue) singularities, are completely removed through virtual recombination within the coarser quantization. In contrast to related methods working as a post-process on the final hexahedral mesh, structure simplifications and the exact preservation of input features are completely decoupled, so both can be achieved.

early step and prevented readjustments, we showed that unfreezing the singular structure leads to improved resolution uniformity and an often preferable simpler block structure of the generated meshes. We furthermore demonstrated in an ablation study that both core ingredients of our approach—flexible structure-preserving constraints and a feature-based objective function—are relevant for achieving these improvements.

Balancing goals While here the main objective was reaching a desired mesh resolution, while keeping feature distortion as low as possible under this condition, a different balancing of these often opposing aspects of simplicity and low distortion can also be highly interesting. Employing a two-step quantization process to that end, like we did for the block-structuring experiments (the first to determine a simple block layout, the second for matching the desired mesh resolution), is also commonly employed in quad meshing [Lyon et al. 2021a,b], but involves educated guessing of parameters that determine the balance of layout coarseness and quality only indirectly. From the perspective of an end-user it would be preferable to have more intuitive and predictable control over this.

Singularity positions While our method is able to simplify the singular structure by removing and recombining singularities, it does not adjust the object space positions of the singularities that remain. Especially when significant simplifications were performed, these positions are often suboptimal for overall map distortion. While they can be optimized as part of a subsequent hex mesh optimization, a more principled approach would be addressing this already on the parametrization level, via an optimization that shifts singularities to minimize the underlying parametrization distortion, as has been done for surface singularities [Campen and Kobbelt 2014]. The added complexity of the volumetric setting, especially regarding the interplay of

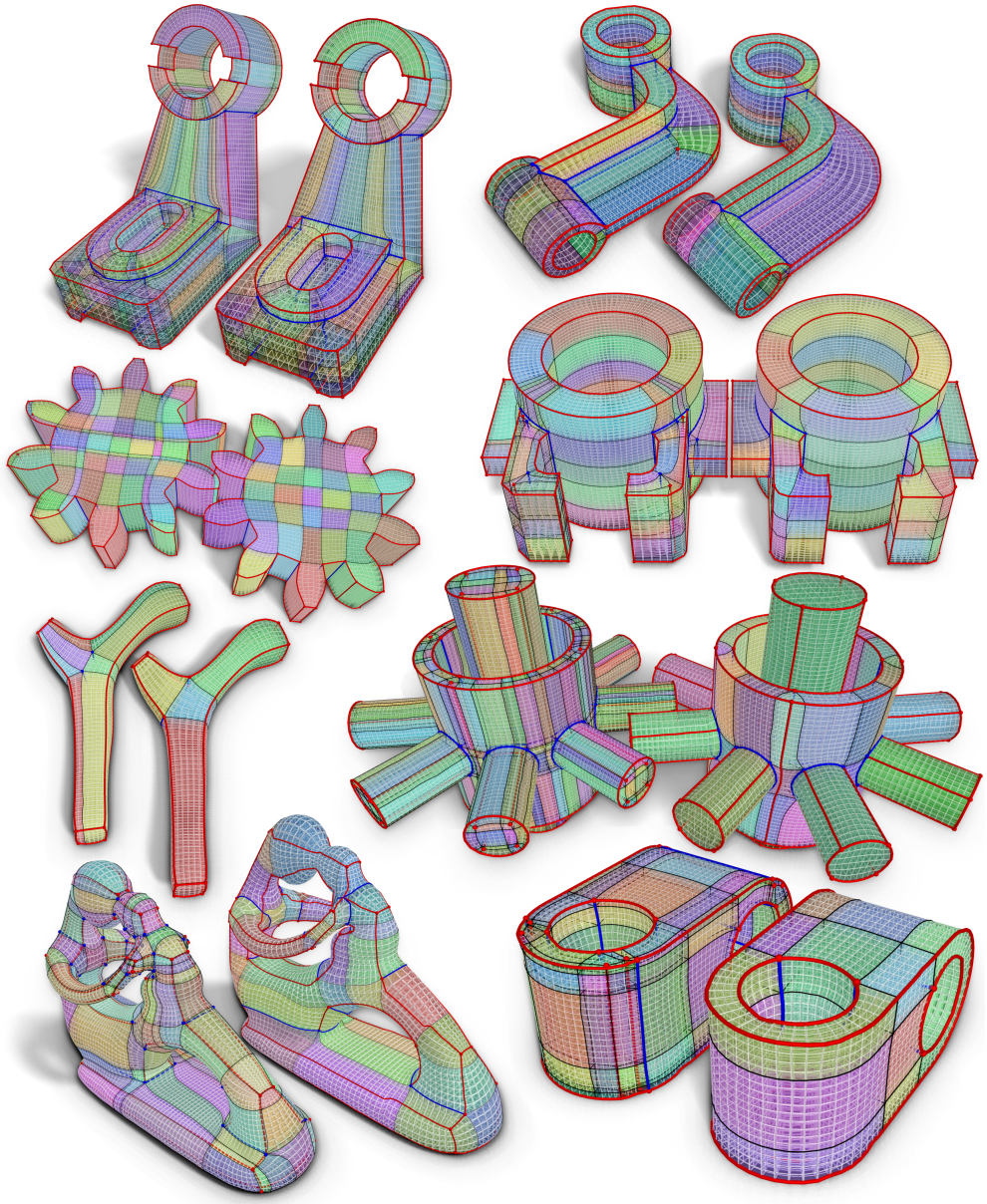


Figure 10.8: Comparison of the base complex of hex meshes generated using $q_{\text{arc,fix}}$ from chapter 7 (left) and the novel $q_{\text{feat,flex}}$ (right) on the same inputs. Valence 3 singularities shown in red, valence 5 in blue.

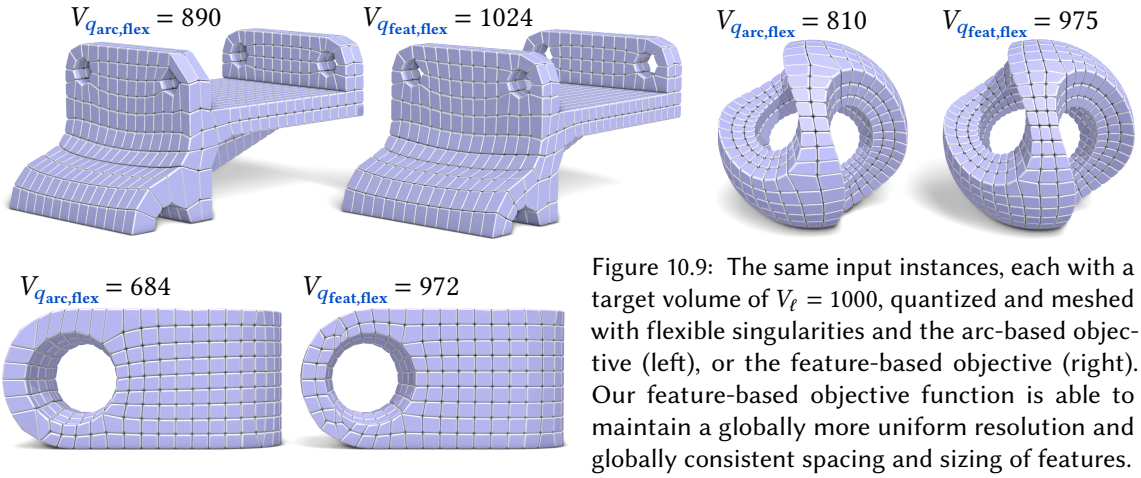


Figure 10.9: The same input instances, each with a target volume of $V_\ell = 1000$, quantized and meshed with flexible singularities and the arc-based objective (left), or the feature-based objective (right). Our feature-based objective function is able to maintain a globally more uniform resolution and globally consistent spacing and sizing of features.

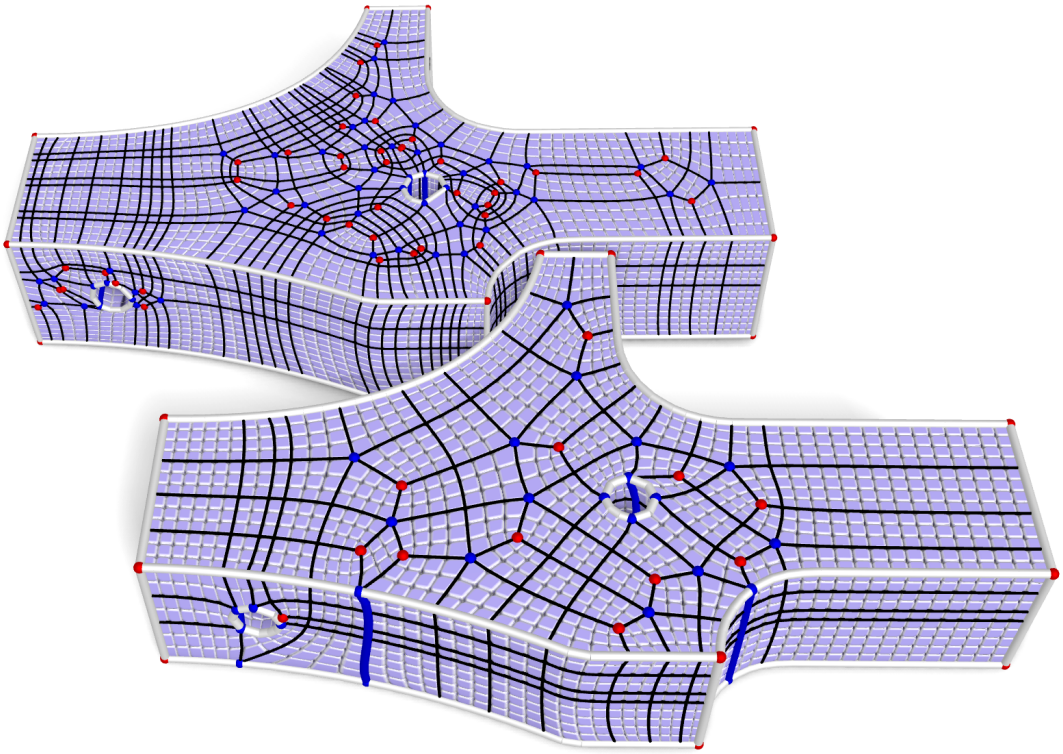


Figure 10.10: On the same input instance, block layouts achieved by the coarsest quantizations with fixed (top) or flexible singularities (bottom). The number of non-feature singularities (red and blue) is significantly reduced from 86 to 27 by our novel quantization. Still, a single quantization and collapsing iteration even for the coarsest target does not yield the theoretically minimal set of singularities in the shown example, due to the (necessarily) *conservative* formulation of singularity index constraints (section 10.2.3.2) preventing the merging of singularities into invalid clusters.

continuous positions and discrete mesh embedding and the general difficulty of robust volumetric parametrization, make this an interesting task for future work.

Solution cut-off The linear structure-preservation constraints of the core integer quadratic program are conservative: they cut off not only the invalid solutions but also some valid ones. This slight over-restriction of the solution space is common to recent quantization methods, but is more obvious for our constraints that prevent singularity clusters with problematic indices; our constraints always force a problematic cluster to split rather than to possibly recombine with another cluster. While this choice is safe in terms of validity, it sometimes leads to suboptimal results in areas where the input singularity structure is particularly messy. Figure 10.10 demonstrates an example for which a significant simplification of the singularity structure can be achieved (reducing 86 non-feature singularities to only 27 in the output), but the resulting output is still not truly minimal. Exploring different branches induced by alternative linear constraints or opting for more general non-linear constraints could improve this. A straightforward solution, if the goal is only to simplify the singular structure, would be to employ multiple iterations of quantizing and collapsing singularities. This can be expected to converge to a truly minimal singularity structure, but is affected by performance slowdowns in the collapsing stage.

Performance Another indirect consequence of allowing flexible singularities, is that that the coarsest possible quantizations often imply a transformation of the T-mesh that approaches or

reaches the lower coarseness limit for cuboidal decompositions of the given domain topology. Transforming the T-mesh from its initial state into this configuration is still guaranteed to be possible via the collapse operators from chapter 9, but in such cases often becomes impractically slow, due to excessive refinement of the underlying tetrahedral mesh to accommodate the twisted T-mesh.

While this previously proved no problem for the maximum coarseness available with singularities fixed, in the current setting much coarser T-meshes become available, requiring longer and more complex sequences of collapses that lead to much more refinement. This refinement leads to the T-mesh collapsing and reparametrization stages taking upwards of 24h on some instances. Note that this is a limitation of only the collapsing step, which can be addressed by more efficient global use of the existing degrees of freedom in the background mesh, rather than greedily creating new ones through local refinement.

The quantization step itself took at most ≈50 minutes. However, this moderate upper bound is in part due to the time limits configured for the IQP solves; in several cases, when the solver was cut off, it still reported a significant optimality gap (up to ≈ 50%). Setting no time limit results in multiple instances also exceeding run times of 24h. Such problematic worst-case behaviour is typical of branch-and-cut integer solvers; we address this issue in the following chapter by introducing a special-purpose solver tailored specifically to our problem type.

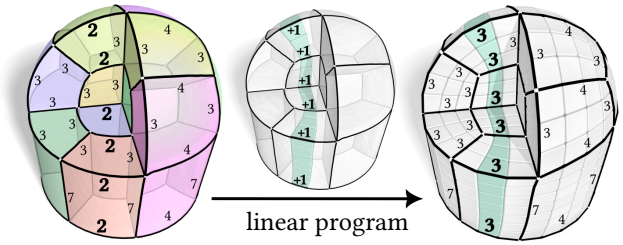
11 Replacing Integer Solvers

IVa–b …embedded T-mesh $\mathcal{T}$ aligned to ϕ

IVc Quantize $\mathcal{T}$ via generic integer solvers

⇕

IVc Quantize $\mathcal{T}$ via continuous LPs



Existing reliable methods for volume quantization are based on solving a sequence of *integer quadratic programs* (IQP). Solving these in a timely and predictable manner with general-purpose solvers proves a challenge, even more so in the open-source field. We observed in previous chapters that a commercial solver performs well on this task in the average case. However, there are also outliers where run time is quite impractically high. Furthermore, the required time is very unpredictable, as there is limited correlation between problem size and required time. Therefore, for practical purposes, additional early-stopping criteria for the solver need to be set (a time limit and/or optimality gap threshold), and choosing these can be challenging. Even more importantly, we observe that open source solvers show significantly lower performance on this task also in the average case. We propose a special-purpose strategy to efficiently compute near-optimal guaranteed-valid quantizations. Effectively: *A minor amount of quality is traded for a major gain in speed and predictability, while retaining full reliability.*

Key to this is that our method does not require a complex integer quadratic programming (IQP) solver, but instead relies on solving a series of continuous *linear programs* (LP). In our formulation, such LPs are used to determine where inflation or deflation of virtual hexahedral sheets are favorable. The benefit is a significantly reduced and more predictable run time, using only open source components, at an often marginal cost in terms of quantization quality.

Our approach is to start with the zero-quantization $q = \mathbf{0}$, assigning zero length to all arcs, which is trivially consistent and non-negative, but of extremely low quality and, more importantly, not structure-preserving. Our LP-based routine then repeatedly performs minimal consistency-preserving modifications of this initial quantization. These modifications, termed *integer-sheet inflations/deflations*, can roughly be thought of as virtually inserting or collapsing sheets of hexahedra in the (not yet existing) hexahedral mesh (cf. fig. 11.1). This is only a partial interpretation, though, as our integer-sheets are more expressive, due to being defined on the non-conforming T-mesh rather than a conforming hex mesh or block layout. Based on the iterative inflation and deflation of integer-sheets, we term the novel T-mesh quantization approach *integer-sheet-pump* (ISP).

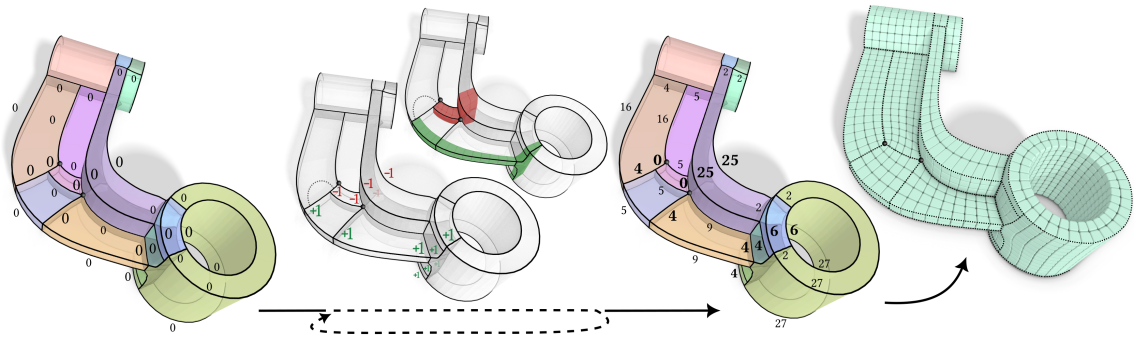


Figure 11.1: Our method solves the quantization problem by starting from a trival initial quantization (left), and applying incremental updates (center left) that improve either constraint satisfaction or the objective function, until a satisfactory solution (center right) is achieved, that corresponds to a valid hex mesh (right). Its central ingredient are integer-sheet inflation/deflation operators (center left), executed on the integer arc lengths of the T-mesh.

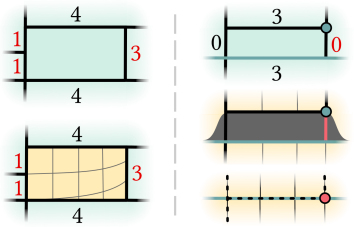
11.1 Integer-Sheet-Pump Algorithm

Problem statement The core T-mesh quantization problem was shown to have the following form (cf. section 7.2):

$$\begin{aligned} \min \quad & d(q,\ell) && && \text{(11.1a)}\\ \text{s.t.} \quad & \mathcal{A}q = \mathbf{0} && \text{(consistency)} && \text{(11.1b)}\\ & \mathcal{B}q \geq \mathbf{0} && \text{(non-negativity).} && \text{(11.1c)}\\ & Cq \geq \mathbf{1} && \text{(structure preservation)} && \text{(11.1d)} \end{aligned}$$

For $d(q,\ell)$ only quadratic distance functions of the form $d_{\mathcal{D}}(q,\ell) = ||q-\ell||^2_{\mathcal{D}} = (q-\ell)^{\mathsf{T}}\mathcal{D}(q-\ell)$ have been explored so far, with $\mathcal{D}$ being either the identity matrix (section 7.2.1) or a matrix approximating feature distortion (section 10.3), making the overall problem of finding $q \in \mathbb{Z}^n$ an *integer quadratic program* (IQP). Notably, for either variant, $q = \ell$ is a continuous optimum.

Consistency constraints require each patch to remain rectangular under the quantization, while structure preservation constraints requires that features and domain topology are preserved. Examples for each an inconsistent and a non-structure-preserving quantization, with their implied integer-grid lines, are shown in the inset. Consistency is required for a quantization to imply a quadrilateral (hexahedral) subdivision of patches (blocks), structure-preservation prevents map defects (or, equivalently, feature collapse). Because ensuring structure preservation is costly to achieve *a priori*, the IQP is solved multiple times, with matrix C augmented between iterations, until a truly feasible solution is found.



Our central goal is to implement a specifically tailored solver for eq. (11.1) in the quantization summarized in algorithm 10.1. We design an algorithm that initializes the quantization to satisfy (11.1b)+(11.1c), then greedily works towards minimizing (11.1a) while preserving (11.1b)+(11.1c) and establishing satisfaction of (11.1d) in the process.

High-level concept Starting from an initial trivial quantization $q^0 = \mathbf{0}$ we obviously have $\mathcal{A}q^0 = \mathbf{0}$ and $\mathcal{B}q^0 \geq \mathbf{0}$. We then wish to perform incremental additive integer-valued updates Δq preserving consistency, i.e., $\mathcal{A}\,\Delta q = 0$. We reject potential updates that would violate $\mathcal{B}(q+\Delta q) \geq \mathbf{0}$. Regarding structure-preservation, however, we need not only updates Δq that maintain but

also ones that *establish* structure-preservation, incrementally reducing the number of violations detected in a previous solution.

Ideally we would like to find an update Δq that respects all these requirements while directly leading to the globally optimal solution, i.e., minimizing (11.1a). But this, of course, is just as hard as solving (11.1). Hence we instead commit to a greedy search approach. Among locally minimal updates Δq that are consistency-preserving and nonnegativity-preserving, we select one that improves the objective (11.1a) as much as possible, or one that satisfies an additional constraint from (11.1d) while deteriorating the objective as little as possible. Choosing such small updates as the steps of our algorithm has two rationales: They can be constructed efficiently, and they enable fine-grained navigation towards the optimum.

Outer optimization algorithm These considerations lead to the high-level view of our algorithm outlined in algorithm 11.1. After initialization of q, the outermost loop alternatingly repeats the inner optimization loop and structure preservation step until global structure preservation is achieved. The inner optimization loop goes through all arcs and tries to find minimal updates bringing arcs closer to their target length, starting with the arcs furthest away from their target. In this way the arc variables q_i that are furthest from their continuous optimum are tackled first. Note that if $\mathcal{D}$ is the identity, this corresponds to $\arg\min_{q_i} \frac{\partial ||q-\ell||^2_{\mathcal{D}}}{\partial q_i}$; otherwise this is not necessarily true.

Algorithm 11.1:
INTEGER-SHEET-PUMP QUANTIZATION

Input: T-mesh with target lengths ℓ
Output: Final quantized integer arc lengths q
// Consistency,non-negativity,objective //
setup matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{D}$
// Structure preservation //
$C \leftarrow$ empty matrix
$q \leftarrow \mathbf{0}$
repeat
  // Greedy optimization //
  while still improving **do**
    $Q \leftarrow$ queue of arcs a_i ordered by $|q_i - \ell_i|$
    while Q not empty **do**
      $a_i \leftarrow Q$.pop()
1     $\Delta q \leftarrow$ MINIMALARCUPDATE(a_i)
2     $t \leftarrow$OPTIMALINTSTEP($\Delta q, \mathcal{B}, C, \mathcal{D}$)
      if $||q + t\Delta q - \ell||^2_{\mathcal{D}} < ||q - \ell||^2_{\mathcal{D}}$ **then**
        $q \leftarrow q + t\Delta q$
        foreach nonzero row i of Δq **do**
          Q.update(a_i)
  // Lazy structure preservation //
  $C' \leftarrow$ LAZYSTRUCTURALCONSTRAINTS(q)
  if C' empty **then return** q
  $C \leftarrow \begin{bmatrix} C \\ C' \end{bmatrix}$
3   $q \leftarrow q +$ MINIMALFEASIBILITYUPDATE(C')

OPTIMALINTSTEP($\Delta q, \mathcal{B}, C, \mathcal{D}$) (algorithm 11.1) computes the optimal integer multiplicity $t \in \mathbb{N}$ for the determined minimal consistent update Δq. The continuous optimum of t^* is given analytically as

$$\begin{aligned} t^* &= \arg\min_t ||q + t\Delta q - \ell||^2_{\mathcal{D}} \\ &= \frac{\Delta q \mathcal{D}(\ell - q)}{\Delta q \mathcal{D} \Delta q}, \end{aligned} \tag{11.2}$$

which can then be rounded to the closest integer satisfying inequality constraints $\mathcal{B}$ and C, yielding the integer optimum due to the objective being symmetric around t^*. This is merely for efficiency; one could always use $t = 1$. For convergence and avoidance of local minima (especially if $\mathcal{D}$ is not the identity) it proves beneficial to employ step sizes between $\frac{t^*}{2}$ and t^*, corresponding to a damping factor common to similar greedy optimization approaches. We concretely perform full steps if $\mathcal{D}$ is the identity, and employ a damping factor of $\frac{3}{4}$ otherwise.

If the found update improves the objective, we apply it and update all affected arc priorities accordingly. Lastly, after convergence of an optimization round we search for violated structure preservation constraints (sections 7.3 and 10.2), returning if there are none. Otherwise we append the new constraints to C and apply a minimal update that re-establishes constraint satisfaction.

Both the arc-based and constraint-driven updates (algorithm 11.1) should fulfill our previously stated requirements: Local minimality, preservation of consistency and non-negativity. Ones of

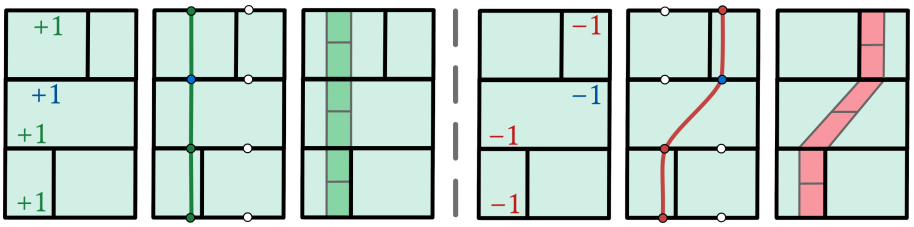


Figure 11.2: Two examples of positive (green) and negative (red) integer-strips Δq constrained to pass through different root arcs (blue) , illustrated as dual paths through the T-mesh and as corresponding (inserted or collapsed) quad strips. Note that not all integer-strips can be simply interpreted as quad strips, see fig. 11.3.

the former type should additionally offer improvements to the current objective value. In the following we focus on how to efficiently find exactly such updates.

11.2 Minimal Consistent Updates

Regarding the construction of locally minimal updates, we can take inspiration from related work on the 2D quantization problem for surface quad mesh generation [Campen et al. 2015]—at least on a conceptual level. In that work, the concept of generating vectors is used to represent minimal quantization updates. In this context, generating vectors are integer vectors Δq representing the smallest consistent building blocks that can form any consistent quantization by incremental addition, when starting from $q^0 = \mathbf{0}$.

The main difficulty lies in determining such favorable generating vectors. Generating vectors Δq fulfilling $\mathcal{A}\,\Delta q = \mathbf{0}$ turn out to correspond directly to certain dual paths in the 2D/surface T-mesh. More specifically, these paths have to always traverse patches end-to-end and have to be cycles or end at the boundary. The crossed arcs have a value of 1 in Δq, all others 0. fig. 11.2 shows an example. Due to their path nature, a simple Dijkstra-type algorithm can be employed to find them. To this end, strictly positive weights are assigned to each arc a_i of the decomposition (and their dual arcs) that gauge how much closer or further $\Delta q_i = 1$ would bring that arc towards its target ℓ_i. Using this, a minimum-weight path constrained to pass through at least one root arc a^* will pass through as few (in a weighted sense) as possible other arcs. Hence, the corresponding vector Δq is a greedily chosen, locally minimal consistent update.

11.2.1 Sheet/Strip Metaphor

Intuitively, such a generating vector (or *integer-strip*) can be thought of as inserting a single additional strip of quads in the quad mesh that is structurally implied by the quantization. This is a limited interpretation, though, as these virtual insertions do not translate one-to-one to incremental updates of the implied quad mesh by classical quad strip insertion (or collapse) operations. Integer-strip insertions are more expressive than quad-strip insertions in the quad mesh, in the sense that a single integer-strip insertion would require multiple (and arbitrarily many) operations on the quad strips of a quad mesh, as demonstrated in fig. 11.3.

This strip metaphor can be lifted to the volumetric setting: positive updates Δq correspond to the number of additional virtual sheets crossing the arcs within the block decomposition, as shown in fig. 11.1 (center). We call those *integer-sheets* and hence our algorithm, repeatedly inflating or deflating them, an *integer-sheet-pump* (ISP).

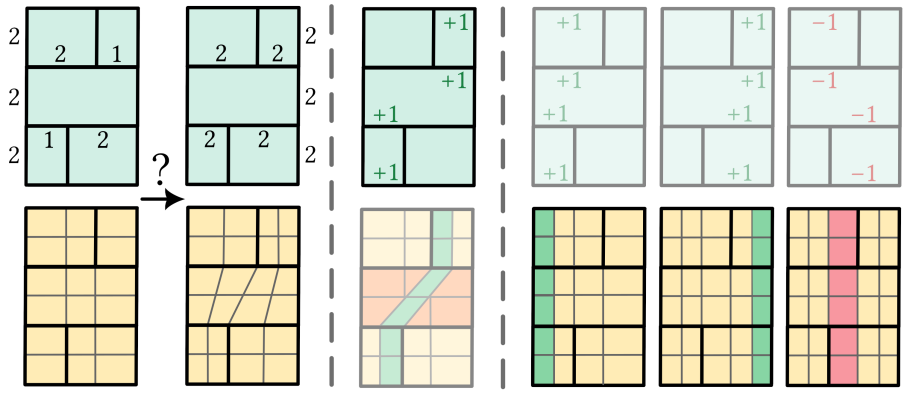


Figure 11.3: Top: quantizations. Bottom: corresponding quad meshes. The transition shown on the left can be achieved through modification of the quantization by the integer-strip shown top center. Note that it has no corresponding interpretation as a quad strip insertion in the mesh. Instead, it rather corresponds to a combination of multiple quad strip insertions and collapses, as shown on the right, and thus is able to capture more complex transitions.

11.2.2 Formulation as a Linear Program

While the determination of integer-strips is equivalent to finding paths in a graph, this concept does not extend to the volumetric setting. Instead, determining an integer-sheet in the cell complex is roughly equivalent to finding a discrete dual *surface* within a polyhedral mesh. This problem has been shown to be expressible as a linear program [Grady 2008].

Essentially, integer-sheets (and also integer-strips in 2D) rooted at an arc a_j are solutions to a linear program of the following form:

$$\min_{\Delta q \geq \mathbf{0}} \quad \sum_{a_i \in A} \Delta q_i w_i \tag{11.3a}$$

$$\text{s.t.} \quad \mathcal{A}\,\Delta q = \mathbf{0} \tag{11.3b}$$

$$\Delta q_j \geq 1 \tag{11.3c}$$

with w_i being the previously mentioned, positive-valued unfavorability weights of arcs. These weights (detailed in eq. (11.5)) represent how unfavorable it is for a strip/sheet to cross a given arc, and the objective function therefore tries to minimize the total unfavorability of the strip/sheet. j is the index of an arc a_j which the strip/sheet is constrained to pass through. Without (11.3c), $\Delta q = \mathbf{0}$ would be the trivially optimal solution; the constraint effectively pumps up the otherwise flat sheet. In the 2D case of integer-strips, $\mathcal{A}$ has a structure that allows the LP to be easily solved via Dijsktra's algorithm. Namely, $\mathcal{A}$ has at most two nonzero entries per column—a result of each arc being incident to at most two patches. In 3D arcs are generally incident to an arbitrary number of patches and thus there are commonly more than two entries per column, preventing this simple approach.

11.2.2.1 Integer Solutions

The above continuous LP always has a rational solution and one such solution is reliably retrieved by a standard simplex algorithm. This follows from the constraint matrices $\mathcal{A}, \mathcal{B}$ and C as well as the constraint right-hand sides only having integer entries [Papadimitriou and Steiglitz 1998]. This rational solution turns out to even be an integer solution in the vast majority of cases. Intuitively, this is due to the objective function trying to push all variables down to their lower bounds—which is an integer 1 for the root arc—while the consistency constraints propagate this

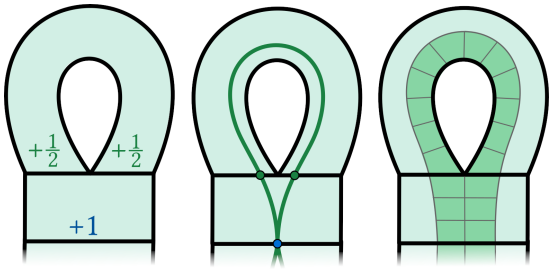


Figure 11.4: 2D example of a T-mesh where constraining the blue arc to value 1 will yield a consistent solution with half-integers. Multiplication by 2 yields an integer-sheet in this case (right).

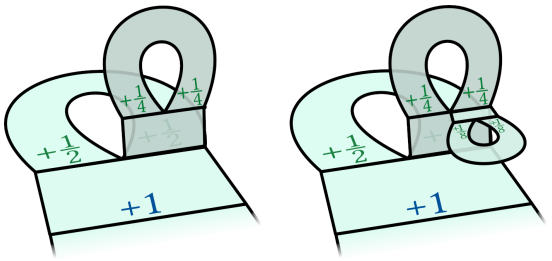


Figure 11.5: 3D example analogous to fig. 11.4. For clarity only some patches are shown, no blocks; imagine an extruded version of this for a complete example. The third dimension allows nesting the U-turn pattern, here illustrating that denominators 2, 4, 8 can result.

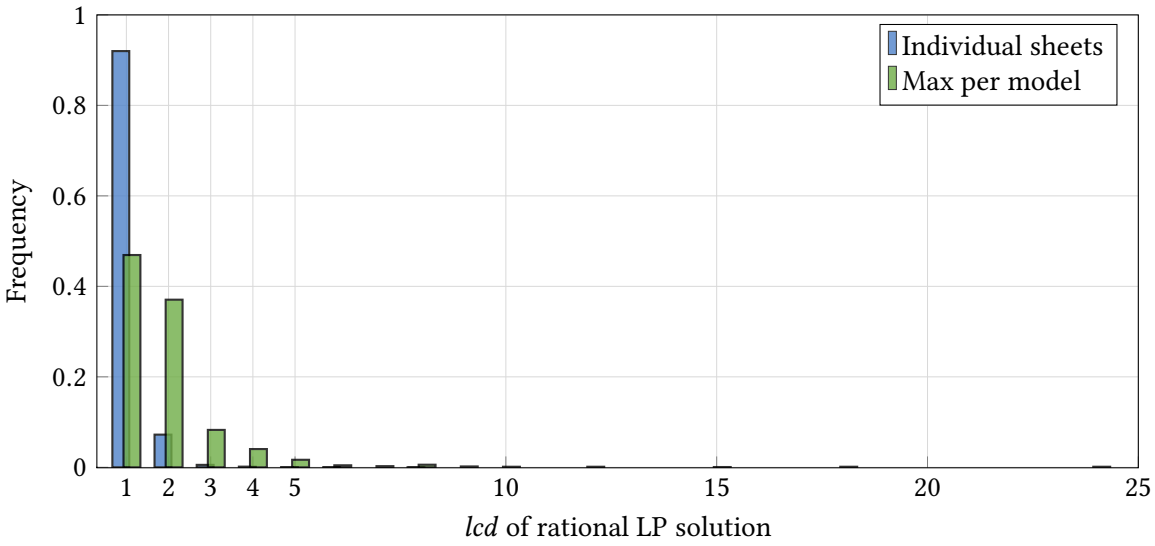


Figure 11.6: Distribution of least common denominators of all the rational LP solutions computed over all models in our test dataset (cf. section 11.4). In 92% of the cases it is 1, i.e., no multiplication is required to yield an integer-sheet.

value across patches without *splitting it up*. Only due to rather intricate interplay of T-junctions, non-integer but rational solutions occasionally occur.

In 2D only one structural non-integer configuration exists [Campen et al. 2015], displayed in fig. 11.4. In this case, going from the rational LP solution to the respective integer-strip simply requires multiplication by factor 2. In 3D, by contrast, non-integer rational solutions with more varied denominators can occur. This is illustrated in fig. 11.5. By further nesting this structural pattern, examples with arbitrarily high denominators could be constructed, although this is unlikely to occur in practice. In any case, an integer solution can be obtained through multiplication with the *least common denominator*.

Note, however, that such an integer solution obtained through multiplication might not actually be locally minimal anymore. We could resort to solving the LP as an actual ILP (with integer-valued variables) in such cases. However, we determined that the lcd differs from 1 quite rarely, and in these cases it is by far most often just 2; see fig. 11.6. It hence appears reasonable to accept this minor local suboptimality for the sake of efficiency and simplicity within the anyway approximative algorithm.

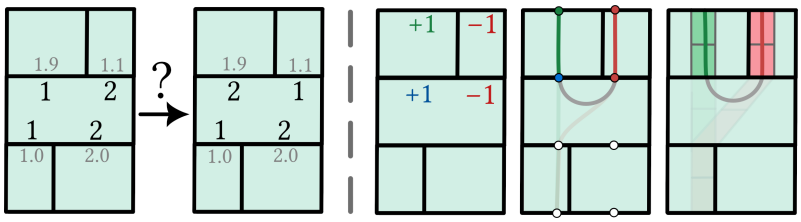


Figure 11.7: The transition depicted on the left is desired, it brings the integer lengths closer to the target values (grey). Via sequential inflation or deflation of integer-sheets, however, this transition cannot be achieved in a monotonic manner. But a *mixed-sign sheet*, which can be interpreted as a sum of multiple sheets, here one inflating (green) and one deflating (red), enables the transition atomically.

11.2.2.2 Weights

To determine integer-sheets Δq whose inflation $(q + \Delta q)$ is favorable, the weights $w_i > 0$ per arc need to be chosen to reflect unfavorability of arc inflation. To determine integer-sheets whose deflation $(q + (-\Delta q))$ is favorable, they need to be chosen to reflect unfavorability of arc deflation.

A weighting scheme that fulfills these requirements is the one also used for the 2D Dijkstra formulation [Campen et al. 2015]. For each arc $a_i \in A$ the weights are built from the signed difference ε_i of each arc's current quantized length from its target length:

$$\varepsilon_i = \begin{cases} \ell_i - q_i, & \text{if inflating} \\ q_i - \ell_i, & \text{if deflating} \end{cases} \tag{11.4}$$

Based on this, a 3-tier hierarchical weighting is employed:

$$w_i = \begin{cases} 1/(\varepsilon_i + 1), & \text{if } 1 \leq \varepsilon_i \\ |A|/(\varepsilon_i + 1), & \text{if } 0 \leq \varepsilon_i < 1 \\ |A|^2(1 - \varepsilon_i), & \text{if } \varepsilon_i < 0 \end{cases} \tag{11.5}$$

Arcs in the highest tier would be shifted towards their target length and consequently have very low unfavorability weights between 0 and 1/2. Arcs in the second tier would be shifted across their target length, so an inflation would be either slightly favorable or slightly unfavorable. These are assigned weights between $|A|$ and $|A|/2$, which by construction is strictly greater than the combined weights in tier 1. The third tier concerns arcs that would be shifted even further away from their target, and are assigned weights at $|A|^2$ and above—larger than the last two tiers combined. This hierarchical weighting ensures that arcs for which a shift in length would be unfavorable will only be shifted if there is no way to shift instead any number of other arcs for which the shift would be favorable.

11.2.2.3 Mixed-Sign Sheets

So far we have silently assumed the variables in our LP formulation to be non-negative. This corresponds to either an integer-sheet inflation or a deflation (when the negated result is added).

It is not hard to see that there are cases for which both simple sheet inflations and deflations fail to escape local minima of the global objective function $d_{\mathcal{D}}(q, \ell) = ||q - \ell||^2_{\mathcal{D}}$. One such case in a 2D setting is shown in fig. 11.7. Here, if $\mathcal{D}$ is the identity the global optimum is not attainable with a greedy algorithm that only considers Δq with non-negative (or non-positive) entries and requires monotonic decrease of $d_{\mathcal{D}}(q, \ell)$. This is a limitation also affecting the above mentioned 2D quantization algorithm [Campen et al. 2015].

Instead of compromising the greedy nature of the approach to overcome this, we suggest here a way of modelling mixed-sign updates $\Delta q = \Delta^+ q - \Delta^- q$, i.e., mixed inflation and deflation, with the same formalism via two non-negative vectors $\Delta^+ q$, $\Delta^- q$. To this end we extend LP (11.3) as follows:

$$\min_{\Delta^+ q, \Delta^- q \geq \mathbf{0}} \quad \sum_{a_i \in A} \Delta^+ q_i w_i^+ + \Delta^- q_i w_i^- \tag{11.6a}$$

$$\text{s.t.} \quad \mathcal{A}(\Delta^+ q - \Delta^- q) = [\mathcal{A} \;\; | -\mathcal{A}] \begin{bmatrix} \Delta^+ q \\ \Delta^- q \end{bmatrix} = \mathbf{0} \tag{11.6b}$$

$$\begin{cases} \Delta^+ q_j \geq 1 \;\wedge\; \Delta^- q_j = 0 & \text{if inflating } a_j \\ \Delta^+ q_j = 0 \;\wedge\; \Delta^- q_j \geq 1 & \text{if deflating } a_j \end{cases} \tag{11.6c}$$

where w_i^+ and w_i^- are the inflation/deflation weights computed via eqs. (11.4) and (11.5), and a_j is an arc constrained to be either inflated or deflated. With this the transition in fig. 11.7 can be achieved in a single greedy step.

11.2.2.4 Enforcing Separation and Non-Negativity

Of our constraints (eqs. (7.1b) to (7.1d)) only consistency has been taken into account so far. Incorporating the other two is straightforward through the relation $q = q_{\text{pre}} + \Delta q$ where q_{pre} is the current quantization before applying the update. Using this substitution, separation and non-negativity constraints can be added to the LP as:

$$C\Delta q \geq \mathbf{1} - C q_{\text{pre}} \tag{11.7a}$$

$$\mathcal{B}\Delta q \geq -\mathcal{B} q_{\text{pre}} \tag{11.7b}$$

These constraints operate in two ways during our greedy optimization. If the current quantization q_{pre} is already separating (and non-negative), then these constraints merely provide bounds for the integer-sheet-pump driven by single arc inflations or deflations (via (11.6c)). When demanding arc deflation, the LP may even become infeasible, signalling that forced deflation of a certain arc would be incompatible with separation or non-negativity. This mode is used in MINIMALARCUPDATE from algorithm 11.1. On the other hand, if the current quantization is currently non-separating—which may be the case initially due to separation constraints being added lazily—no arc-driven pumping via (11.6c) is needed; the sheet is *self-inflating* via (11.7a). This mode is used in MINIMALSEPARATINGUPDATE from algorithm 11.1. Both algorithms are listed in section 11.3.1 for reference.

Failsafe fallback A challenge that presents itself here are the occasional non-integer, rational solutions to the LP. If one avoids these by re-solving the LP with integer constraints as an ILP, there is no problem. Otherwise, if an integer-sheet is obtained via lcd-multiplication, additional care needs to be taken. The rational quantization update by construction would stay within all bounds but a multiple of it might not. Note, however, that this can only be the case when negative values are involved. Hence, for non-integer updates we encounter that become infeasible through multiplication, we solve restricted LPs instead (with $\Delta^- q = \mathbf{0}$ and negative values suppressed in C). This yields a solution from a more constrained yet feasible space, which can always be scaled as necessary to obtain an integer solution that establishes separation. Details on this fallback are given in section 11.3.1.

Algorithm 11.2:
MINIMALARCUPDATE

Input: Target lengths ℓ; current quantization q_{pre}; matrices $\mathcal{A}$, $\mathcal{B}$, C, $\mathcal{D}$; arc a_j
Output: Integer update Δq inflating or deflating a_j

$\Delta q \leftarrow 0$
if should inflate a_j **then**
  $\Delta q \leftarrow$ solve (11.8a), s.t. (11.8b) to (11.8e)
else
  if (11.8c) or (11.8d) clash with eq. (11.8f) **then**
    return **0**
  $\Delta q \leftarrow$ solve (11.8a), s.t. (11.8b) to (11.8d) and (11.8f)
if Δq not integer **then**
  $\Delta q \leftarrow \text{lcd}(\Delta q) \cdot \Delta q$
  if Δq violates (11.8d) or (11.8c) **then**
    return **0**
return Δq

Algorithm 11.3:
MINIMALFEASIBILITYUPDATE

Input: Target lengths ℓ; current quantization q_{pre}, matrices $\mathcal{A}$, $\mathcal{B}$, C, $\mathcal{D}$,
Output: Integer quantization update Δq enforcing structure preservation

$\Delta q \leftarrow$ solve (11.8a), s.t. (11.8b) to (11.8d)
if Δq is integer **then** return Δq
// Escalate until integer and feasible //
$\Delta q \leftarrow \text{lcd}(\Delta q) \cdot \Delta q$
if Δq violates (11.8c) or (11.8d) **then**
  $\Delta q \leftarrow$ solve (11.8a), s.t. (11.8b) to (11.8d) and (11.8g)
  if Δq is not integer **then**
    $\Delta q \leftarrow \text{lcd}(\Delta q) \cdot \Delta q$
    if Δq violates (11.8d) **then**
      $C \leftarrow C^+$
      $\Delta q \leftarrow$ solve (11.8a), s.t. (11.8b) to (11.8d) and (11.8g)
      if Δq is not integer **then**
        $\Delta q \leftarrow \text{lcd}(\Delta q) \cdot \Delta q$
return Δq

11.3 Final Formulation

With all extensions described in section 11.2.2 the final LP is:

$$\min_{\Delta^+q,\Delta^-q\geq\mathbf{0}} \quad \sum_{a_i\in A} \Delta^+q_i w_i^+ + \Delta^-q_i w_i^- \tag{11.8a}$$

$$\text{s.t.} \quad [\mathcal{A} \;|-\mathcal{A}] \begin{bmatrix}\Delta^+q\\ \Delta^-q\end{bmatrix} = \mathbf{0} \tag{11.8b}$$

$$[\mathcal{B} \;|-\mathcal{B}] \begin{bmatrix}\Delta^+q\\ \Delta^-q\end{bmatrix} \geq -\mathcal{B}q_{\text{pre}} \tag{11.8c}$$

$$[C \;|-C] \begin{bmatrix}\Delta^+q\\ \Delta^-q\end{bmatrix} \geq \mathbf{1} - Cq_{\text{pre}} \tag{11.8d}$$

$$\text{optionally} \begin{cases} \Delta^+q_k \geq 1 \;\wedge\; \Delta^-q_k = 0 & (11.8e)\\ \Delta^+q_l = 0 \;\wedge\; \Delta^-q_l \geq 1 & (11.8f)\\ \Delta^-q = \mathbf{0} & (11.8g)\end{cases}$$

Defining $\varepsilon_i^\pm = \pm(\ell_i - q_i)$, the weights $w_i^\pm$ are given by

$$w_i^\pm = \begin{cases} 1/(1+\varepsilon_i^\pm), & \text{if } 1 \leq \varepsilon_i^\pm \\ |A|/(1+\varepsilon_i^\pm), & \text{if } 0 \leq \varepsilon_i^\pm < 1 \\ |A|^2(1-\varepsilon_i^\pm), & \text{if } \quad \varepsilon_i^\pm < 0 \end{cases}. \tag{11.9}$$

This LP is invoked in multiple ways by algorithm 11.1, via calls to MINIMALARCUPDATE and MINIMALSATISFYINGUPDATE. These subroutines are summarized in algorithms 11.2 and 11.3, respectively. We investigate the final LP structure in more detail in the following, including the aspects of integrality of continuous solutions, fallback solutions (in case the continuous solution is not integral), and speed-ups through straightforward simplifications of the above problem.

11.3.1 Problem Structure Analysis

Integer Solutions Especially important for integer linear programming are problems with a structure that allows cheaper-than-exponential computation of optimal solutions. Specifically

desirable are problems, for which the relaxation to the continuous domain still yields an integer solution (e.g., via the simplex algorithm). Typically, the linear objective $c^\mathsf{T}x$ is assumed variable, putting the focus on criteria for constraint systems $Mx \geq b$ that guarantee an integer solution irrespective of c, so called *totally dual integral* (TDI) systems. Such systems yield a feasible set that is a (high-dimensional) polyhedron $P(M) = \{\boldsymbol{x} \in \mathbb{R}^n \mid M\boldsymbol{x} \geq b\}$ whose vertices are integral. Due to the simplex algorithm traversing the vertices of this polyhedron, it yields an integer optimum (assuming feasibility and boundedness).

A known, sufficient condition for a system $Mx \geq b$ being TDI, even irrespective of b (as long as b is integral), is M being a *totally unimodular* (TU) matrix [Hoffman and Kruskal 2009]. A matrix M is called totally unimodular if the determinant of every square submatrix of M is also from $\{-1, 0, 1\}$. Various equivalent descriptions of total unimodularity are known.

The system of constraints of (11.8) can be brought into canonical form, yielding:

$$\begin{bmatrix} \mathcal{A} & -\mathcal{A} \\ -\mathcal{A} & \mathcal{A} \\ \mathcal{B} & -\mathcal{B} \\ C & -C \\ \mathbf{1}_k^\mathsf{T} & \mathbf{0}^\mathsf{T} \end{bmatrix} \begin{bmatrix} \Delta^+ q \\ \Delta^- q \end{bmatrix} \geq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ -\mathcal{B} q_{\text{pre}} \\ \mathbf{1} - C q_{\text{pre}} \\ 1 \end{bmatrix}, \tag{11.10}$$

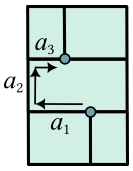
where $\mathbf{1}, \mathbf{0}$ are all-one or all-zero column vectors of adequate size and $\mathbf{1}_k^\mathsf{T}$ is a unit row-vector with a single 1 in position k, such that $\mathbf{1}_k^\mathsf{T} \Delta^+ q \geq 1$ is a pump constraint enforcing a single arc a_k to become inflated or deflated. Obviously the system matrix is not generally TU, otherwise all obtained solutions would be integer.

In fact, the counter-examples figs. 11.4 and 11.5 demonstrate that already $\mathcal{A}$ is not TU. Concretely, they show examples of arcs that are on the same side of a patch p, yet appear on the opposite sides of a straight chain of patches (forming a cycle with p). Within $\mathcal{A}$ this corresponds to them appearing on at least one row with the same sign (both entries are $+1$ or -1), while appearing in another row (explicitly or after row transformations) with opposite signs. This describes exactly a 2×2 submatrix $C_{\text{sub}} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ of $\mathcal{A}$ (or of an equivalent row-transformed version of it) with determinant ± 2, which precludes $\mathcal{A}$ being TU.

Based on our observations, we conjecture that in the absence of such cycles, the global system matrix is actually TU (or at least TDI). Such cycles are evidently quite rare in practice, limiting both the frequency and the 'severity' (caused by cycle nesting) of non-integer solutions. Lastly, our LP formulation exploits the *locality* of cycles: Minimizing a positive-weight sum over variables favors solutions covering as few variables as possible; hence a problematic cycle that does not lie near the pumped arc is rarely included in the solution.

Failsafe Fallback We derive in the following a safe fallback for cases, where the solution of (11.8) does not yield an integer solution, and the scaled up solution is not feasible. This is specifically relevant for MINIMALSATISFYINGUPDATE, where constraint feasibility must be established rather than preserved. In the LP (11.8) all equality constraints are homogeneous. If in addition all inequality constraints had non-negative coefficients only, any positive multiple of a solution would be a solution as well. This is of relevance for the rare cases in which we need further quantization updates in order to establish separation, but the LP yields a non-integer solution that needs to be scaled up. By using a non-deflating mode in this case, i.e., constraint (11.8g) is activated, all relevant coefficients of eq. (11.8c) are non-negative. The separation constraints (11.8d), however, may contain some negative coefficients in matrix C—if formulated as described in section 7.4. algorithm 11.3 therefore, if necessary, switches to a non-negative matrix C^+ (defined below) to

ensure separation can always be achieved. The separation constraints arise from a graph search on the T-mesh. A path between two features that implies zero quantized distance between these serves as a witness of feature nonseparation. Based on the discovered path, an additional constraint is devised that enforces separation. Such a path, by construction, always has one axis along which it expands monotonically—here the axis of arc a_2. Along the other axes the path may wind back and forth, as seen here for a_1 and a_3. Separation along one axis is sufficient, i.e., the desired constraint is $q_2 > 0 \vee q_1 - q_3 \neq 0$. This can be turned into a linear constraint (for separation matrix C) in a conservative manner in different ways. By default, $q_2 + (q_1 - q_3) > 0$ is used. When building matrix C^+ we instead turn it into $q_2 > 0$, i.e., separation specifically along the monotone axis is asked for, leading to only non-negative coefficients.



Simplifications In the complete LP formulation, stated in section 11.3, there are two variables per arc, two equality constraints per patch, and a varying number of inequality constraints. We observe that often not all variables are mutually dependent via constraints. This allows partitioning the problem into multiple independent subproblems, which can be solved more efficiently. To this end, we determine the clusters of variables that are transitively dependent via constraints. For each cluster then a separate LP with those constraints that involve the respective variables is formed. On average this resulted in around 5 subproblems in our experiments.

Furthermore, we can trivially reduce the problem size of these subproblems, by eliminating one variable and one constraint for each equality constraint that involves only two variables, i.e., for each pair of opposite patch sides consisting of only one arc each. This situation occurs quite frequently in the T-meshes studied here: only around 15% of consistency constraints involved more than two variables in our experiments.

Reference implementation We have implemented our method as a modification of our open source library QGP3D discussed in section 7.5.1. It employs CLP from the *COIN-OR* project [Lougee-Heimer 2003] as an open-source solver for linear programs. The QGP3D library can be be built with any combination of these solvers: Generic commercial IQP [Gurobi Optimization, LLC 2025], generic open-source IQP [Bonami et al. 2008], or ISP based on a generic LP solver [Lougee-Heimer 2003]. Users can then explicitly choose and configure the employed solvers via the library API.

11.4 Results

Experiments Experiments were conducted on a modern 32-core server CPU. While multi-threaded performance might be an interesting subject to investigate, our experiments solely focus on single-threaded performance, allocating exactly one thread per experiment instance.

An experiment instance here is an input model equipped with a seamless parametrization (based on which an aligned T-mesh can be computed) and a scaling factor. Target lengths of arcs in the constructed T-mesh are given by their length under the seamless parametrization, scaled by aforementioned global factor. This factor effectively controls the resolution of the resulting hexahedral mesh. As input we use the same datasets as employed in section 9.6; the T-mesh is once more computed via the motorcycle complex algorithm from chapter 5, configured to perform only regular-reductions. Each model we use in combination with 8 different scaling factors, chosen per model such that 0%, 1%, 5%, 10%, 17%, 25%, 50% or 100% of arcs have a target length below 0.5, respectively, yielding a total of 1568 test instances. Spreading out the scaling

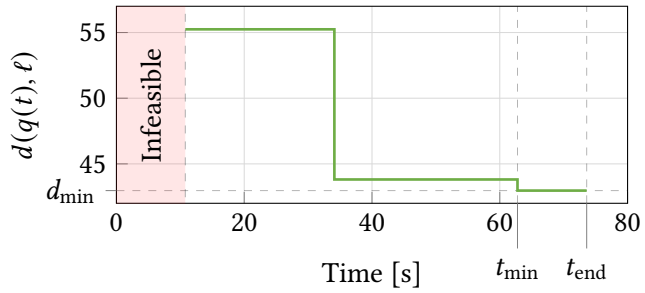


Figure 11.8: Generic example of solver behavior over time. Branch-and-bound IQP solvers as well as our ISP method are (after an initial phase) *anytime algorithms*, i.e., can be interrupted to return intermediate results (of increasing quality over time).

factors like this, we aim to cover the full range of potential application scenarios ranging from the generation of maximally coarse block layouts to dense hexahedral meshes, where the optimization problems might behave quite differently.

We compare our ISP method to two implementations of the IQP formulation from chapter 7 based on different general purpose solvers as back-ends, using the arc-based objective and fixed singularities. Our main point of comparison is using the open-source solver BONMIN [Bonami et al. 2008]. Additionally, we report timings for using the commercial solver GUROBI [Gurobi Optimization, LLC 2025]. A short extension of our evaluation that covers the flexible-singularity setting described in chapter 10 (chronologically published after our here discussed work on the ISP solver) is given in section 11.4.3.

11.4.1 Quantization Evaluation

Comparison methodology In this part of our evaluation we want to focus on three key aspects to gauge our method's performance against IQP formulations based on general-purpose solvers. Those are execution speed, predictability of run time and optimality of results obtained within certain time frames.

All three aspects are not as straightforward to compare between implementations as one may expect. Often, general-purpose integer solvers may find a good or even optimal solution after a moderate amount of time and then spend a disproportionately larger amount of time on closing the optimality gap—not necessarily by improving the solution but by incrementally determining that no better solution exists. In the worst case this involves exploring an exponential amount of promising but eventually worse alternative solutions. When comparing execution times, it is not immediately obvious which time to use for comparison—the one where a (in some sense) good solution was found, the one where the optimal solution was found, or the final time when the solver terminates.

Hence, in the following we compare not a single time and a single objective value but rather entire time curves of the objective values $d(q(t), \ell)$. An example of such a time curve is shown in fig. 11.8. Until a solution that is feasible with respect to eqs. (7.1b) to (7.1d) is encountered, the objective value can be considered undefined or—more practically—infinite. The second particular point of interest is the final drop of the curve, i.e., the time of acquisition of the final solution and its objective value, which we refer to as $t_{\min}$ and $d_{\min}$, respectively. Lastly, the final termination time we denote by t_{end}. These curves and their key indicators form the baseline for the following evaluations.

Optimality over time For this evaluation we aim to find each method's "average time curve" of the objective value, over the entire set of test instances. Because both the time and objective scale may differ by orders of magnitude, we need to normalize both axes. For the objective value we normalize via division by the (quasi-)optimum, the final objective $d_{\min}$ of the commercial IQP

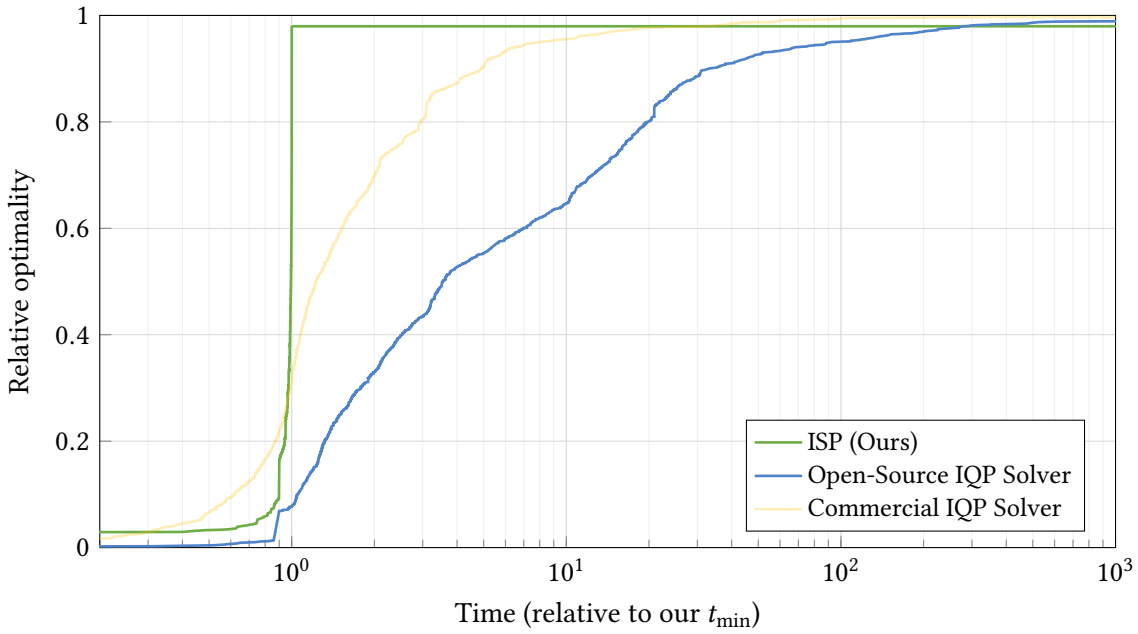


Figure 11.9: Comparison of relative optimality over time on a per-model basis. Plots of all test instances were normalized along the time axis, relative to $t_{\min}$ of our algorithm, then averaged. Our method achieves its average final optimality of ≈98% in about a hundredth of the time the open-source IQP solver needs to reach similar optimality levels on average and in about a tenth of the time the commercial IQP solver requires. Past these times both IQP solvers achieve minor further improvement on average, albeit at diminishing rates.

solver version (with only a generous time limit of 5h). Because the result will be in the range $[1, \infty)$ we take the inverse before averaging the resulting curves. This inverse then represents a relative optimality between 0 (infeasible) and 1 (quasi-optimal). For the time axis, to make comparison between our method and others clearest, we normalize via division by $t_{\min}$ of our method. The result of this normalized averaging is displayed in fig. 11.9. Evaluation of the plot suggests that for the average problem instance our algorithm is most likely to achieve near-optimal results at 98% optimality, and perform faster than both the commercial and open-source IQP methods by one or two orders of magnitude, respectively.

To demonstrate the performance of our method when processing a large dataset sequentially, we measured a second time series in which all experiment instances are processed sequentially and both time and relative optimality are accumulated over the course of the experiment. In this case, running the IQP solvers without an early termination criterion is impractical, as they would sometimes spend huge amounts of time to marginally improve or to conclude optimality before advancing to the next test instance. For fair comparison, we therefore test our method against the IQP approach with early termination. Concretely, in one experiment we allow termination once the solver's optimality gap drops below 30%, in a second experiment we instead set an upper time limit, enabling early termination after 20 seconds of processing time per model. The resulting cumulative optimality curve is plotted in fig. 11.10. Remarkably, the gap-based stopping criterion, despite allowing a rather high suboptimality of 30%, proves to be very ineffective at reducing the overall execution time of the IQP solver. On the other hand the time-limited IQP solver performs objectively worse than our approach in terms of both speed and optimality. We conclude that our method in this scenario again performs clearly better in terms of speed and comparably in terms of optimality, while not requiring the tuning of meta-parameters like early termination criteria to achieve this.

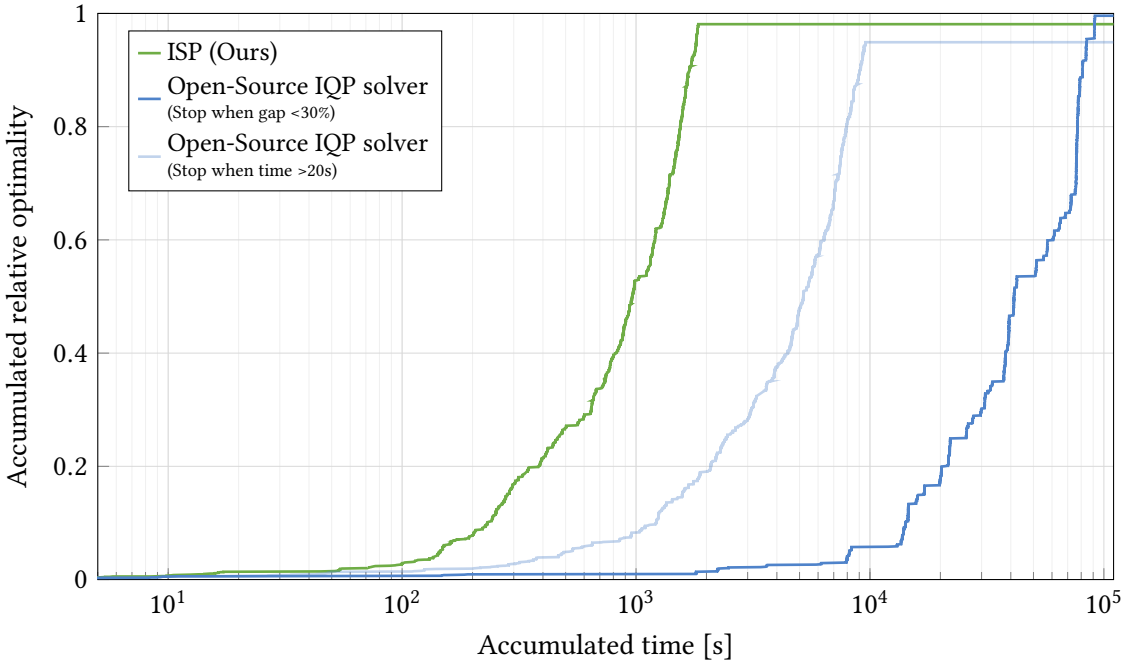


Figure 11.10: Comparison of relative optimality over time for sequential processing of the whole dataset. The y-axis represents the normalized sum over model-wise relative optimalities. Our method finishes processing all 1568 instances in just 30 minutes. Despite the large allowable optimality gap of 30%, the total processing time of the gap-limited IQP solver exceeds ours by two orders of magnitude, showing the ineffectivity of this early stopping criterion. The time-limited IQP solver performs about an order of magnitude faster than its gap-limited counterpart but drops to 95% overall relative optimality, thereby being surpassed by our algorithm in terms of both speed and quality.

Predictability of t_{end} Generally, a desirable property of an algorithm is allowing to roughly estimate the necessary execution time to produce useful output based on the complexity of the input. For the quantization scenarios relevant here, a simple measure representing input complexity is the number of arcs in the input cell decompositions, because these are directly associated with the variables of interest.

Figure 11.11 shows the distribution of algorithm termination times t_{end}. For the IQP solvers a 10% optimality gap was allowed as early termination condition. Overall, the qualitative differences in run time are reflected here once more, with our method in the lowest tier of termination times. More interestingly, though, we see wildly diverging termination times for models above ≈300 arcs in case of the open-source IQP solver or above ≈800 arcs for the commercial solver, in some cases spanning a large range of run times even for the same, but differently scaled input model. Due to the time limit that was necessary for practicality, it is unclear how much further this divergence actually goes for large inputs. For our algorithm, run times seem to scale much more predictably with input size.

Optimality of d_{min} We reported above already the average accuracy of our method when compared to a (quasi-)optimal benchmark. fig. 11.12 shows a more detailed view of the distribution of final objective function values d_{min}. For the IQP approach we consider two values per instance: the final objective value and the (possibly intermediate) objective value at 10× the time t_{end} required for our algorithm to terminate.

Halting the IQP solver after 10× the time required, results in an equally good objective value for 71% of instances, in a better value in about 12% of cases and a worse value in 17% of cases,

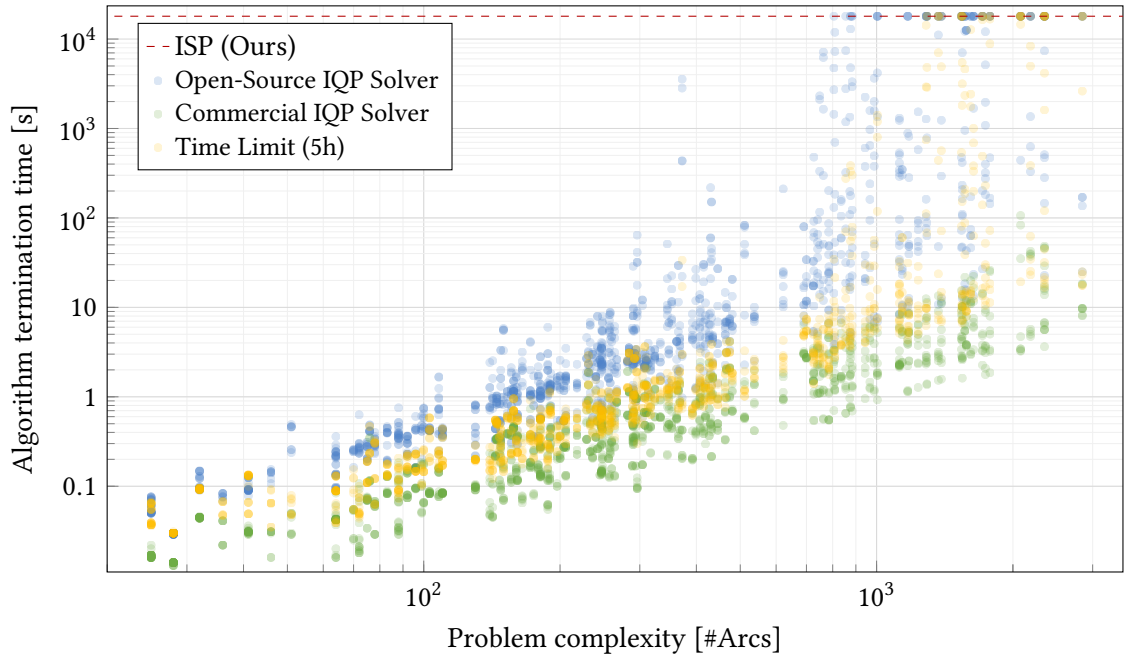


Figure 11.11: Comparison of run time over input size, as represented by the number of arc variables. Run time variance gives an estimate of the predictability of algorithm run times. IQP solvers, due to large gaps between typical and worst-case performance, are typically lacking in this regard—as becomes evident also here, with variances of up to four orders of magnitude for similarly complex inputs. Run times of our algorithm on similar inputs rarely differ by more than one order of magnitude and never by more than two.

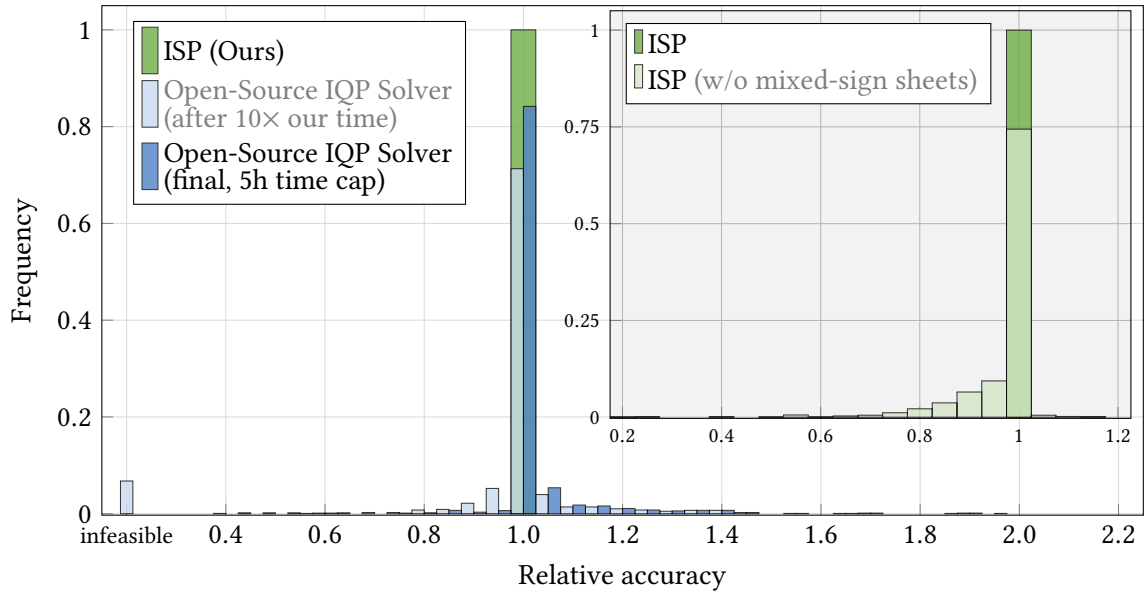


Figure 11.12: Comparison of other methods' accuracy (in terms of achieved objective value) relative to our method, over the test instances. It can be seen that even after granting the IQP solver 10× the time per instance that our method takes, it has not found an even just feasible quantization for about 7% of the instances, and the quantization found for the others is slightly worse than ours on average. Furthermore, our method's result is close to optimal or optimal in the vast majority of cases: The final solution returned by the IQP solver after nearly unlimited time (a generous time cap of 5h) is better by 2.6% on average. Regarding the few outliers, see fig. 11.14. The inset illustrates the benefit of enabling mixed-sign sheets. Notice how in some cases it is crucial in order to escape bad local minima.

including 7% where no feasible solution had been found at all. Our final solution in turn was on par with the final solution of the IQP approach in 84% of cases, worse in 16% of cases and better in 2% of cases. Such latter cases, where our greedy algorithm outperforms the IQP algorithm in terms of optimality, can occur for two reasons: first, the time limit of 5h for the IQP solver, and second, the fact that the lazily added separation constraints [Brückler et al. 2022a] are over-zealous (sufficient but not necessary); depending on the path of optimization, different parts of the feasible space are excluded.

Significance of mixed-sign sheets Finally, let us also evaluate the significance of our proposed extension to mixed-sign sheets. As demonstrated in fig. 11.7 we have identified cases, where considering only uniform-sign updates leads to getting stuck in certain local minima—even in 2D cases. The inset of fig. 11.12 demonstrates the performance of our algorithm with and without mixed-sign sheets enabled. Enabling mixed-sign sheets yields significantly better final objective values in about a quarter of cases, with best-case improvements by factors of 2 and above. Let us remark that these mixed-sign sheets are a general conceptual improvement also over the greedy 2D quantization algorithm [Campen et al. 2015]; such mixed-sign updates are not amenable to that algorithm's Dijkstra-based approach.

11.4.2 Hexahedral Meshing

Hexahedral meshes implied by the computed quantizations were produced using the pipeline from section 8.3: The quantization on the T-mesh is used only to determine exactly the integer degrees of freedom of a global integer-grid map subsequently optimized for low distortion. The resulting hexahedral sheets extracted from it do not necessarily align with the precursor T-mesh walls, only the marked features and singularities are matched by the mesh as intended.

As a consequence, differences in optimality of a T-mesh quantization do not directly translate into equivalently large differences in hex mesh quality, but rather inaccuracies in achieving exactly the desired target resolution of the hex mesh—especially when the differences in quantization accuracy are relatively small, like with the average 2% suboptimality of our method's results. fig. 11.13 demonstrates to what extent typical differences between our method's quantization and the optimal quantization are reflected in the resulting hex meshes—including a note on the relative speedup for reference. As conjectured before, there are no noticeable differences, not even if the quantization accuracy differs by 10% or 20%. Taking a look at the so-called base complex of the generated hex meshes, one can find small differences in the layout, mostly due to different relative alignment of singularities in the mesh structure.

Outliers There are however, in contrast to the cases discussed above, a handful of situations where the exact alignment of singularities in the mesh play a role also for hexahedral element quality and where the ISP quantization obtains an evidently worse solution. Specifically if there are singularities that in the T-mesh are barely misaligned and confined by other critical entities surrounding them, it can be relevant that the quantization realigns the singularities by very strategically placed 0s. Otherwise there may be very flat or misshaped hexahedral sheets in the output that need to squeeze between singularities and boundaries in geometrically unfavorable ways. fig. 11.14 shows one of the rare examples where our quantization process ends up in a rather suboptimal local minimum that essentially contains an additional sheet—that cannot monotonically be removed using our operators. As it is very flat near the base of the ramp (squeezing between the boundary and a prescribed singularity), it has a strong impact on the

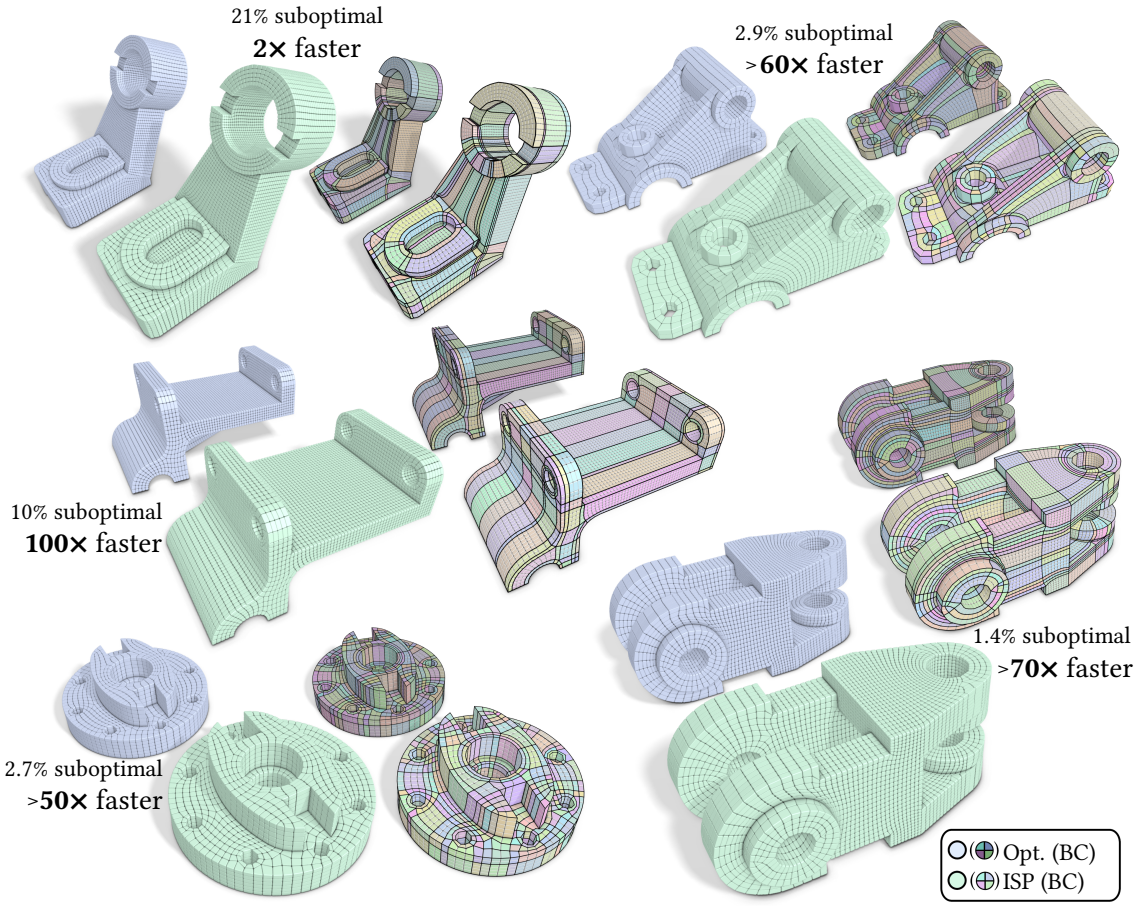


Figure 11.13: A selection of hex meshes produced from optimal quantizations (blue) or from our ISP quantizations (green). Beside the plain hex meshes a highlighting of the structure of the *base complex* (BC) is shown, which allows an easier visual inspection of singularity alignment and element count. Times (relative to the commercial IQP solver) and suboptimality refer to the quantization phase.

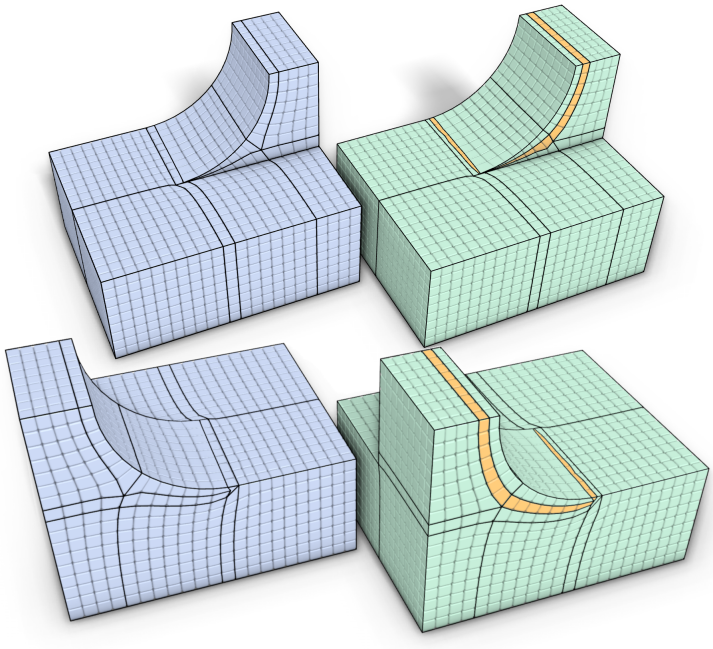
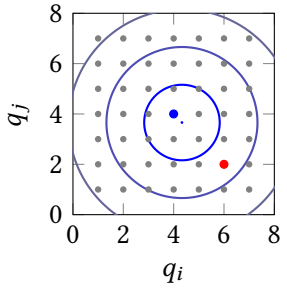


Figure 11.14: Left: result using the IQP solver. Right: result using our ISP method. The main difference is an additional sheet (orange). This causes a quantization objective higher by 57%. In essence, one more layer of hexahedra squeezes into the sharp tip near the base of the ramp, or in terms of the quantization: more very short arcs of target length ≈0 were quantized to length 1. Neither configuration is truly desirable, a better positioning of the squeezed valence 3 singularity at the base of the ramp is of greater importance for quality.

objective value. In part this is also due to the prescribed singularity structure being quite badly located near the base of the ramp in this input example from the dataset.

11.4.3 Feature-Based Objective Function

The integer-sheet pump algorithm in the form described so far is tailored specifically to the arc-based objective (rather than the feature-based objective from section 10.3), i.e., $\mathcal{D}$ has been assumed to be the identity matrix. In that case the energy landscape is isotropic, level sets are (high-dimensional) spheres (circles for pairs of variables, see the inset), and arc variables are independent from one another, objective-wise. Hence, moving each arc closer to its global target is optimal not only globally but also locally. In other words: For any current quantization q the vector pointing towards the global (continuous) optimum $(\ell - q)$ is a scaled version of the negative gradient $-\nabla||q-\ell||^2_{\mathcal{D}}$. Hence, there is only a single notion of favorable update direction, which makes assigning arc variable priorities for the queue in algorithm 11.1, as well as assigning unfavorability weights eq. (11.9) unambiguous. Not as evidently, the greedy algorithm can reach the global optimum (barring constraints) by a chain of single-coordinate descent steps, see the red dot in the inset.

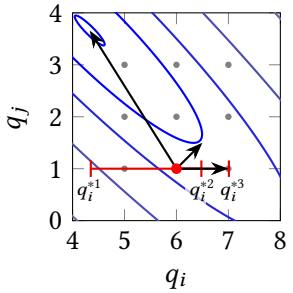


If $\mathcal{D}$ is not the identity, $(\ell - q)$ and $\nabla||q-\ell||^2_{\mathcal{D}}$ are not colinear vectors. For individual arcs a_i, the negative partial derivative $-\frac{\partial||q-\ell||^2_{\mathcal{D}}}{\partial q_i}$ may even point away from the global optimum. Hence, we can define three different notions for evaluating the favorability $\varepsilon_i^{\pm} \pm (q_i^* - q_i)$ of an arc update (positive sign for inflation, negative for deflation):

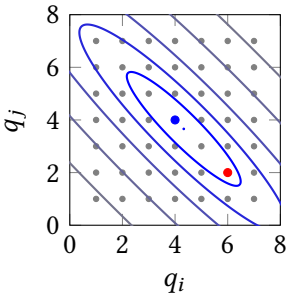
a) q_i^{*1} is the global (continuous) optimum, i.e., ℓ,

b) q_i^{*2} is the optimum of q_i along the gradient direction at the current quantization.

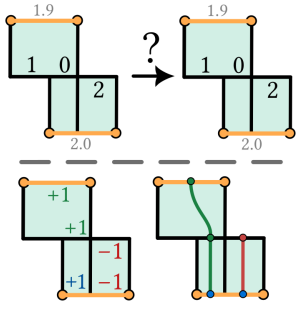
c) q_i^{*3} is the optimum of q_i assuming all other variables fixed at their current values.



Lastly, it is possible, that the greedy optimization gets stuck because it tries only single-variable updates (actively pumping just a single arc). The objective landscape for a general $\mathcal{D}$ has rotated (high-dimensional) ellipsoid level sets. For pairs of variables those are ellipses, potentially rotated, as exemplified in the inset. If the current quantization is the red point, then lowering the objective value requires diagonal updates. Finding this path via greedy updates requires updating two variables at once.



Intuitively, these situations can occur due to arc variables being grouped into inter-feature paths via $\mathcal{D}$ (cf. section 10.3). Sometimes, one path is locally at its optimal integer length, but it would be beneficial (globally) that one arc along that path grows and another shrinks. Doing either individually incurs a temporary optimality cost, hence the greedy algorithm discards both. Doing both at once keeps the important path near-optimal, bypassing the optimality threshold and escaping the local minimum. An example of such a



situation is illustrated in the inset: The top transition is desirable, and can only be achieved in an objective-monotonic manner by pumping two arcs at once, as shown in the bottom part.

The above considerations motivated investigating the following aspects for the setting where $\mathcal{D}$ is not the identity but the feature distortion matrix from section 10.3:

1. Order the arc queue in algorithm 11.1 according to $\varepsilon_i^{\pm}$ based on either of the above.
2. Set the weights (11.9) according to $\varepsilon_i^{\pm}$, based on either of the above, based on a maximum, minimum or average of the above.
3. To escape from local minima, try double- and triple-variable updates (adding two or three pump constraints per LP).

We exhaustively tested different combinations of the above, computing feature-based flexible-singularity quantizations over the whole set of instances discussed in section 10.4.1, for which the default ISP configuration reached an average relative optimality of only 80% (relative to the commercial solver). We determined that a favorable balance between speed and optimality when minimizing the feature-based objective is achieved by the following configuration

1. We order the arc queue based on $q_i^{*1} = \ell$ (same as before, this choice made practically no difference).
2. We employ the same LP weighting scheme as before by default (based on $q_i^{*1} = \ell$); once stuck, we switch to one based on the maximum of $\varepsilon_i^{\pm}$ computed from q_i^{*1} and q_i^{*3}. Experiments showed that the performance of using only one weighting scheme was significantly worse than switching to a fallback, once the default got stuck. The mentioned combination had the best worst-case and average performance out of all (ordered) binary combinations of schemes.
3. We employ only single-variable updates by default; once stuck, try double-variable updates by imposing both (11.8e) and (11.8f) for pairs of arcs that lie in the same inter-feature path. We observed greedy optimization often getting stuck without double-variable updates; however, triple variable updates offered no noticeable benefit in our tests. Note that the complexity of checking pairs only as a fallback, and only for arcs within the same inter-feature path—which cover the T-mesh only sparsely—is sub-quadratic in general.

These modifications of the default ISP method improve the average optimality gap for ISP quantizations in the feature-based flexible-singularity setting from about 20% to 5%—almost matching the $\approx 3\%$ in the arc-based fixed-singularity setting reported in section 11.4. We also double-checked whether any of the above modifications benefit quantization scenarios using the arc-based objective and fixed singularities; this was not the case. The modified ISP (computing flexible-singularity quantizations) is slower than the default ISP (computing arc-based, fixed-singularity quantizations) by a factor close to 2 in all measures (worst-case, average and percentiles), hence still much faster than generic integer solvers as reported in section 11.4.

Part E

Synopsis

If I look back I am lost.

Daenerys Targaryen
in *A Dance with Dragons*
by George R. R. Martin

12 Conclusion

In this thesis we have examined in detail the 3D integer-grid map pipeline, which demonstrably provides the most general and versatile approach to generating hexahedral meshes among recent methods. This pipeline, despite its 2D variant being theoretically well understood and widely used for surface quad meshing, was found to lack theoretical foundations, reliable algorithms, and—by virtue of these issues—widespread practical adoption in the 3D volumetric case. In an effort to bridge these gaps, we have presented several contributions that improve the reliability of the overall pipeline through theoretical guarantees, while simultaneously offering practically *useful* algorithms rather than purely theoretical solutions.

After giving a summary of our contributions, we examine remaining issues and open questions from both a researcher's and a practitioner's perspective, addressing the aspect of practical adoption of our academic results, and pointing out open questions for future research.

Summary

Of the five main steps of the pipeline, fully satisfactory solutions have been known only for steps I and V. For steps II, III and **IV**, only best-effort solutions with relatively high failure rates existed. Recent research has focused largely on finding better solutions for steps II and III. The complexity of the quantization step **IV**, i.e., of transforming a continuously relaxed seamless map into an actual integer-grid map, was mostly disregarded in prior literature, and solutions have been employed for which failures are easily demonstrated.

- I Tet meshing the domain
- II Frame field computation
- III Seamless map computation
- **IV** Seamless map quantization
  - **IVa** Integer fixing
  - **IVb** Integer-grid re-mapping
- V Hex mesh extraction

Research perspective We identified two major challenges in quantization: **IVa** finding valid integer assignments, and **IVb** finding an injective map adhering to the chosen integers. Our main contribution lies in developing data-structures and algorithms revolving around these, in sum overcoming these two challenges and providing fully reliable solutions for quantization.

As data structures, we employed coarse volumetric T-meshes, constructed reliably as *motorcycle complexes* on the seamless map. A theoretical analysis of the construction yielded important guarantees for the obtained T-mesh, namely that it decomposes the domain into map-aligned cuboids, with all features of the background domain carrying over to the T-mesh. To make our theoretical guarantees applicable to real-world data, we extended a numerical sanitization routine for 2D seamless maps to our 3D setting. Beside the envisioned use within our quantization algorithms, these non-conforming, cuboid T-meshes are of independent interest, e.g., for solid T-spline construction.

In our setting, the obtained T-meshes crucially allowed a reduction in the complexity of further processing steps, by reducing problems that scale with tet mesh density to ones that scale only with the topological complexity of the domain, typically several orders of magnitude lower. This complexity reduction enabled the formulation of an efficient algorithm for determining valid quantization integers, by iterative solution of an integer quadratic program. While our initial formulation relied on the straightforward but impractical use of generic integer solvers, we subsequently developed a specialized solver, based on only continuous linear programming, that solves the same problem near-optimally but orders of magnitude faster.

The T-meshes were further instrumental in reliably solving the problem of finding an integer-grid map that not only adheres to the chosen integers, but is also *injective*, i.e., valid. The key insight here was to reduce the global problem of computing a complex-topology volumetric map to the local problem of computing many simple-topology cuboid maps, which can be solved reliably with known methods. To get rid of undesirable T-mesh cells prior to the block-wise mapping, we developed an algorithm for collapsing the cells of embedded T-meshes. The formulation of this problem via generic and robust collapse operators allows such simplifications to be applied to arbitrary cell complexes embedded within a finer mesh; a formalism that applies to any type of volume partition. A first use case for such operators outside the realm of hex meshing has already been pointed out; they enabled a similar partition-and-solve approach for a different class of volume maps [Hinderink et al. 2024].

Revisiting the now fully reliable quantization with a different perspective, under which the pillars of the seamless map—its singularities—are no longer assumed fixed, we extended previous theoretical considerations on the validity of more restricted quantizations, and developed modifications of aforementioned algorithms that guarantee validity even in this more flexible setting.

Practitioner perspective For all of the mentioned algorithms and data structures we published reference implementations in the form of open-source libraries TRULYSEAMLESS3D, MC3D, QGP3D and C4HEX, all publicly available with an unrestrictive license. Such single-purpose libraries, solving only some steps of a larger pipeline of course are of little use to practitioners. For a long time, no public implementations have been available at all for some steps of the pipeline, and existing research prototypes for other steps were often not compatible with each other.

To this end, we integrated our implementations into ALGOHEX, a research-level implementation of the complete integer-grid map pipeline, that incorporates also (best-effort) state-of-the-art methods for steps II and III. By virtue of our contributions, as well as advances on frame field

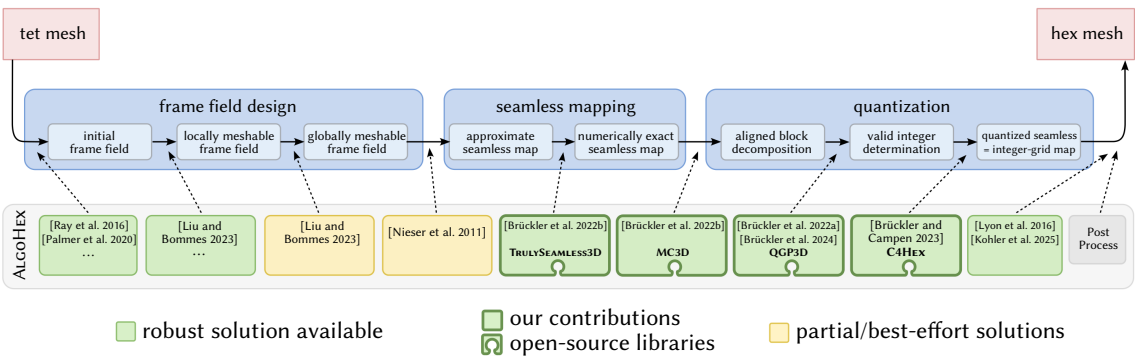


Figure 12.1: Overview of the contributions presented in this thesis in the broader scope of the hex meshing pipeline based on integer-grid maps and its implementation in the ALGOHEX library [Liu and Bommes 2023]

meshability [Liu and Bommes 2023], the success rate of this pipeline on the challenging HEXME dataset [Beaufort et al. 2021] has increased from 10% to 58%. Notably, robustness is unconditional once a seamless map is found, allowing hex meshes of arbitrary resolutions to be generated—even with modified (simpler) topology when employing our flexible-singularity approach.

Open Questions

Meshability The aforementioned 42% failure rate of the most recent implementation of the integer-grid map pipeline primarily stems from the still largely unsolved problem of computing *meshable* frame fields, i.e., ones for which the frame field singularities exactly correspond to valid hex mesh singularities. Already fully solved is only the issue of *local meshability* of frame fields [Liu and Bommes 2023]; this guarantees that the local neighborhood around any point in the domain corresponds to a valid hex mesh. Alas, this does not guarantee that a valid hex mesh exists that can connect all such local neighborhoods. As the naming implies, global frame field meshability requires the fulfillment of global conditions rather than local ones; a formalization of these conditions that leads to a constructive algorithm is still mostly outstanding. Global meshability is practically a non-factor for surface quad meshing: Only a rather specific, easily avoided type of non-meshable cross fields exists [Shen et al. 2022], assuming singularity indices satisfy a simple global sum constraint [Ray et al. 2008]. Hence, there is very little 2D work from which to draw inspiration on how this problem could be solved in 3D.

An—admittedly vague—idea could be to investigate energy formulations on frame fields or singular graphs under which necessary repair operations emerge, so to speak, i.e., to devise an energy in which non-meshability manifests itself in the form of spatially local energy peaks, guiding the application of singularity modifications that lower this energy. Related are 2D methods that solve for a seamless parametrization, with sufficient singularities emerging directly from the problem formulation [Diamanti et al. 2015; Coiffier and Corman 2023]. The argument to be made for such an approach in 3D is that obtaining a valid parametrization could potentially be the best constructive way to obtain guaranteed-valid parametrization singularities. However, the state-of-the-art in volume parametrization is not nearly as advanced as for surface parametrization, and it is unclear whether such methods could be extended to 3D at all.

T-mesh without seamless map Another idea to approach the global meshability problem, or at least come closer to understanding it, could be to try and construct a T-mesh already from the (potentially non-meshable) frame field, prior to seamless mapping. In surface quad meshing, such approaches have been employed successfully [Myles et al. 2014; Pietroni et al. 2016, 2021]. While in 2D, separatrices are curves, they are surfaces in 3D, which are not generally available from purely local frame-field integration. Recent advances in frame-field integrability [Vekhter et al. 2025] could be of help here. However, even if a consistent iso-surface could be traced by on-the-fly frame field integration of local neighborhoods of the surface's front, it is clear that this will not result in a valid global T-mesh for globally non-meshable frame fields. Studying invalid configurations that occur in such cases, however, could be of great value to improve the understanding of global meshability conditions. Potentially, even repair operations for invalid T-mesh configurations and the underlying singular graph become tangible in such a setting. In any case, we believe that investigating an extension of our volume T-mesh tracing routine to frame fields could at least provide valuable insights into the global meshability problem.

In 2D, a more general class of maps, so called *seamless similarity* maps [Campen and Zorin 2017], serve as alternative guiding structures for T-mesh construction that can even be found for

non-meshable orientation fields and singularity configurations. While the T-meshes constructed on such maps do not generally admit a conforming subdivision, they still guarantee an aligned quadrilateral partition, on which modifications recovering meshable singularities by collapsing or adding singularities are conceivable (as outlined in section 7.5.1 or done in [Pietroni et al. 2021, Fig. 2], respectively). To the best of our knowledge, no extension of seamless similarity maps to the volume has been considered in the literature. While it is hard to say for sure whether a 3D generalization of such maps, admitting the construction of an aligned cuboid T-mesh partition, is feasible, and—assuming it is—whether such maps can be efficiently computed in the volume setting, it certainly seems a promising avenue to explore.

Injective volume parametrization While several works guarantee finding injective surface maps even with seamlessness and alignment constraints via an intrinsic convex [Campen et al. 2019, 2021; Shen et al. 2022] or combinatorial [Zhou et al. 2020; Levi 2022, 2023] reformulation of the problem, no such technique exists so far for the volumetric case. A first attempt to generalize the first class of intrinsic reformulation approaches to 3D [Paillé et al. 2015] does not result in a convex problem. Context-agnostic numerical methods in 3D [Du et al. 2020; Garanzha et al. 2021] have been very successful on limited datasets, yet still often fail to obtain an injective map for general parametrization problems. Robust, combinatorial approaches to 3D parametrization show first promising results [Campen et al. 2016; Hinderink and Campen 2023; Nigolian et al. 2023; Hinderink et al. 2024; Nigolian et al. 2024], but are still limited in terms of generality and efficiency [Meloni et al. 2024]; only basic scenarios and constraints are supported.

In this class of methods, a variation of our embedding modification operators has been used for generalizing reliable (cut-free) volume mappings from star-shaped domains [Hinderink and Campen 2023] to arbitrary shapes of ball-topology [Hinderink et al. 2024], by enabling the compatible partition of a simplicial map's domain and codomain. Extending this to topologically more complex domains, while seeming feasible, is elusive due to the intricacies in compatibly decomposing arbitrary volumes into topologically simple subvolumes.

Adding seamlessness and alignment constraints is a whole different issue. A promising direction to investigate in this regard could be to leverage the fact that *maintaining* the highly non-convex injectivity constraint is easier than *establishing* it after the fact. Rather than performing numerical untangling of a non-injective but otherwise constraint-satisfying map, one might initialize the map injectively (e.g. as the identity) ignoring all other constraints, and then optimize for reduced constraint violation while carefully maintaining injectivity. Research in this direction could draw from similar approaches to polycube mapping [Huang et al. 2014a; Fang et al. 2016; Mandad et al. 2022], in which a non-polycube map is progressively transformed into a map satisfying polycube constraints. While convergence towards full constraint satisfaction is still uncertain in this reverse approach (even for polycube maps), in practice convergence could very well be better as other constraints (besides injectivity) are typically convex and hence easier to satisfy from an infeasible starting point.

Singularity Structure Optimization The above considerations only cover complete failures to produce output; another aspect to consider are situations where a hex mesh is produced, but of unsatisfactory quality. In integer-grid map based hex meshing, mesh quality is largely determined by the quality of the intermediate maps, which in turn can be severely impaired by a suboptimal singular structure, stemming from a bad (yet meshable) frame field.

Clearly, though, once a valid seamless map has been established, this can offer a more convenient basis for improving an initial singular structure than a frame field. For one thing, map singularities exactly correspond to hex mesh singularities—which is not the case for frame

fields—avoiding explicit meshability concerns. Additionally, a seamless parametrization induces a coarse T-mesh of the domain via our motorcycle complex algorithm, which is a convenient proxy structure on which to globally coordinate modifications. Once a modification plan has been devised, our general-purpose embedding operators can be used to execute it, with topological guarantees.

Our flexible-singularity approach demonstrates the potential of such an approach to some extent, but has several limitations. A major one is the inability to freely relocate singularities, as only collapsing or merging of existing singular edges is supported; this is discussed further below. The other limitation is that the simplification of the singular graph, while significant, is generally not maximal: It typically can not cancel all possible pairs of singular edges, due to (necessarily) conservative quantization constraints. On the datasets available to us, this was rarely an issue; singular structures were relatively clean to begin with.

However, it is not unusual that the singularity graphs determined from frame fields are much more complex than necessary even if meshable (see e.g., Liu and Bommes [2023, Fig. 23]), especially for complex inputs. It would be interesting to further investigate ways to more aggressively simplify the singular structure after a seamless map is already available. As an ad-hoc solution—probably not the most efficient one—one could simply perform multiple iterations of our flexible-singularity method, alternating between computing a T-mesh quantization and collapsing zero-cells from it. Due to singularities being the *cause* of overly conservative constraints in the approach, constraints become less conservative in each iteration, allowing more and more simplification. Another approach would be to adjust the constraints so that they do not preclude the merging of certain singularities one desires to cancel out, but potentially even enforce this. Achieving this, while maintaining the feasibility and termination guarantees of our method could be an interesting avenue for future work.

Singularity Position Optimization While singularities could be optimized as part of a subsequent hex mesh optimization, a more principled approach would be addressing this already on the parametrization level. Ideally, singularity relocation could be incorporated into the map optimization itself; then, singularities could be moved around as needed to improve map quality, measured by an energy function over the piecewise-affine map. Usually, optimization schemes for such energies assume only the parameters (codomain) to be variable, and the object space (domain) to be constant. Relocating singularities, however, requires some movement in object space—at the very least by shifting the singularity "tag" around discretely, but ideally also allowing local continuous deformations for better convergence. Hence, parameters of vertices but also positions of at least some vertices (at and around singularities) should be optimized in a coupled manner.

For surface singularities, such an approach has already successfully been employed [Campen and Kobbelt 2014]. It encounters two core challenges: Firstly, it must properly handle the co-dependency of discrete singularity tags and continuous vertex positions and parameters. This is done by computing optimal shifts only for vertices tagged as singularities, but not applying this update to the vertex but to the singularity tag. As this singularity tag would then end up on a position that does not coincide with a vertex, a close-by vertex is snapped onto the new singularity position and tagged as singular. Connectivity-wise, this corresponds to the singularity tag moving along a chain of edges, which must be tracked to maintain secondary constraints.

The second challenge is finding a practical optimization scheme for the highly non-linear and non-convex energy that results even for relatively simple parametrization-related energies, when assuming both the domain and codomain of the piecewise affine map as variable. To this end, the authors employ an alternating approach, assuming either domain or codomain as constant in

subsequent iterations and computing descent steps only for the other. In conjunction, this results in a rather efficient approach that demonstrably finds good surface singularity positions.

It appears promising to extend this approach to 3D seamless maps. Such an endeavor requires solving the above challenges for the arguably harder case of network-like singularities and tet-based maps. As the optimization is still vertex-based, but singularity tags are edge-based, this raises two questions: How to translate singular vertex updates to singular edge updates, and how to compatibly update entire singular links at once. For this task, it appears that our consistency-preserving embedding operators could be quite useful. Even when assuming this aspect settled, finding an energy formulation and working out a matching optimization scheme that efficiently navigates the energy landscape without getting trapped in local minima, seems a serious challenge motivating future research. In terms of handling energies with no or hard-to-derive analytical derivatives, recent domain-specialized frameworks for automatic differentiation [Schmidt et al. 2022] are likely of great practical value.

The Author's Publications

I

The contents of this thesis are largely based on the following prepublications, in which the author of this thesis was the main (first) author.

H. Brückler, O. Gupta, M. Mandad, and M. Campen. 2022b. "The 3D Motorcycle Complex for Structured Volume Decomposition." ***Computer Graphics Forum*****, 41, 2**
The second and fourth author had the leading roles in the conceptualization and writing of an original draft, with the third author contributing the initial concept and draft of the sanitization routine. A proof-of-concept implementation was provided by the second author for the tracing routine and by the third author for the sanitization. The author of this thesis joined the project at an early stage, sharing a leading role with the fourth author from there, contributing major parts of the core methodology, most of the implementation, experimental validation, evaluation, and visualizations, as well as large parts of the final submission.

H. Brückler, D. Bommes, and M. Campen. 2022a. "Volume Parametrization Quantization for Hexahedral Meshing." ***ACM Transactions on Graphics*****, 41, 4**
The author of this thesis and the third author took the leading roles in conceptualization and writing of the original draft, with the second author joining for valuable discussions and to design experiments, together leading to breakthroughs in developing the full methodology. The implementation, experimental validation & evaluation, as well as result visualizations are attributed to the author of this thesis. The final version was written jointly with the third author, with reviewing contributions by the second author.

H. Brückler and M. Campen. 2023. "Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing." ***ACM Transactions on Graphics*****, 42, 6**
The author of this thesis took the leading role in conceptualization, writing of the original draft, and developing the final methodology, supported in parts of these aspects by the second author. The implementation, experimental validation, evaluation and visualization are attributed to the author of this thesis. For writing and reviewing the final version the author of this thesis took the leading role, supported in parts by the second author.

H. Brückler, D. Bommes, and M. Campen. 2024. "Integer-Sheet-Pump Quantization for Hexahedral Meshing." ***Computer Graphics Forum*****, 43, 5**
The author of this thesis was responsible for conceptualization and writing of the original draft. The second and third authors assisted in developing the full methodology. The implementation, experimental validation, evaluation and visualization are attributed to the author of this thesis. For writing and reviewing the final version, the author of this thesis took the leading role, supported in parts by the third author, with reviewing contributions by the second author.

H. Brückler and M. Campen. 2026. "Volume Quantization with Flexible Singularities for Hexahedral Meshing." ***Computer Graphics Forum*****, 45, 2**
The author of this thesis was responsible for conceptualization and writing of the original draft. The second author contributed the initial idea for the feature-based objective. The implementation,

experimental validation, evaluation and visualization are attributed to the author of this thesis. The author of this thesis wrote and reviewed the submitted version, assisted during review by the second author.

Bibliography

F. Alauzet and D. Marcum. 2014. "A closed advancing-layer method with changing topology mesh movement for viscous mesh generation." In: *Proceedings of the 22nd international meshing roundtable*. Springer, 241–261.

M. Alexa. 2020. "Conforming weighted Delaunay triangulations." *ACM Transactions on Graphics*, 39, 6, 1–16.

M. Alexa. 2019. "Harmonic triangulations." *ACM Transactions on Graphics*, 38, 4, 1–14.

P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. 2005. "Variational tetrahedral meshing." In: *ACM SIGGRAPH 2005 Papers*, 617–625.

M. Alnæs et al.. 2015. "The FEniCS project version 1.5." *Archive of numerical software*, 3, 100.

Altair Inc. 2025. *HyperMesh*. https://www.altair.com/hypermesh/. (2025).

R. Anderson et al.. 2021. "MFEM: A modular finite element methods library." *Computers & Mathematics with Applications*, 81, 42–74.

ANSYS Inc. 2025. *ANSYS*. https://www.ansys.com/products/meshing. (2025).

F. Aqilah, M. Islam, F. Juretic, J. Guerrero, D. Wood, and F. N. Ani. 2018. "Study of mesh quality improvement for CFD analysis of an airfoil." *IIUM Engineering Journal*, 19, 2, 203–212.

C. G. Armstrong, H. J. Fogg, C. M. Tierney, and T. T. Robinson. 2015. "Common themes in multi-block structured quad/hex mesh generation." *Procedia Engineering*, 124, 70–82.

D. Arndt et al.. 2023. "The deal. II library, version 9.5." *Journal of Numerical Mathematics*, 31, 3, 231–246.

K. Barrett. 1996. "Jacobians for isoparametric finite elements." *Communications in numerical methods in engineering*, 12, 11, 755–766.

M. Baumeister and L. Kobbelt. 2023. "How close is a quad mesh to a polycube?" *Computational Geometry*, 111, 101978.

P.-A. Beaufort, M. Reberol, H. Liu, F. Ledoux, and D. Bommes. 2021. "Hex Me If You Can." *arXiv preprint arXiv:2111.10295*.

L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. 2014. "The hitchhiker's guide to the virtual element method." *Mathematical models and methods in applied sciences*, 24, 08, 1541–1573.

S. E. Benzley, E. Perry, K. Merkley, B. Clark, and G. Sjaardama. 1995. "A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis." In: *Proceedings, 4th international meshing roundtable*. Vol. 17. Sandia National Laboratories Albuquerque, NM, 179–191.

M. Bern, P. Chew, D. Eppstein, and J. Ruppert. 1995. "Dihedral bounds for mesh generation in high dimensions." *contract*, 14, 88-K, 0591.

K. Y. Billah and R. H. Scanlan. 1991. "Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks." *American Journal of Physics*, 59, 2, 118–124.

T. Blacker. 2000. "Meeting the challenge for automated conformal hexahedral meshing." In: *Proceedings of International Meshing Roundtable*, 11–20.

T. D. Blacker. 1996. "The cooper tool." *5th International Meshing Roundtable, 1996*.

T. D. Blacker and R. J. Meyers. 1993. "Seams and wedges in plastering: a 3-D hexahedral mesh generation algorithm." *Engineering with computers*, 9, 2, 83–93.

D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, and L. Kobbelt. 2013. "Integer-Grid Maps for Reliable Quad Meshing." *ACM Transactions on Graphics*, 32, 4, 98:1–98:12.

D. Bommes, T. Lempfer, and L. Kobbelt. 2011. "Global structure optimization of quadrilateral meshes." *Computer Graphics Forum*, 30, 2, 375–384.

D. Bommes, H. Zimmer, and L. Kobbelt. 2009. "Mixed-integer quadrangulation." *ACM Transactions on Graphics*, 28, 3, 77:1–77:10. ISBN: 978-1-60558-726-4.

D. Bommes, H. Zimmer, and L. Kobbelt. 2010. "Practical mixed-integer optimization for geometry processing." In: *International Conference on Curves and Surfaces*. Springer, 193–206.

P. Bonami et al.. 2008. "An algorithmic framework for convex mixed integer nonlinear programs." *Discrete Optimization*, 5, 2.

M. J. Borden, S. E. Benzley, and J. F. Shepherd. 2002. "Hexahedral Sheet Extraction." In: *Proc. 11th International Meshing Roundtable*, 147–152.

J. Born, P. Schmidt, and L. Kobbelt. 2021. "Layout embedding via combinatorial optimization." In: *Computer Graphics Forum* 2. Vol. 40, 277–290.

M. Botsch and L. Kobbelt. 2004. "A remeshing approach to multiresolution modeling." In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 185–192.

M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. 2010. *Polygon mesh processing*. CRC press.

X. Bourdin, X. Trosseille, P. Petit, and P. Beillas. 2007. "Comparison of tetrahedral and hexahedral meshes for organ finite element modeling: an application to kidney impact." In: *20th International technical conference on the enhanced safety of vehicle, Lyon*.

A. Bowyer. 1981. "Computing dirichlet tessellations." *The computer journal*, 24, 2, 162–166.

M. Bracci, M. Tarini, N. Pietroni, M. Livesu, and P. Cignoni. 2019. "HexaLab. net: An online viewer for hexahedral meshes." *Computer-Aided Design*, 110, 24–36.

J. Bronson, J. A. Levine, and R. Whitaker. 2013. "Lattice cleaving: A multimaterial tetrahedral meshing algorithm with guarantees." *IEEE transactions on visualization and computer graphics*, 20, 2, 223–237.

H. Brückler, D. Bommes, and M. Campen. 2024. "Integer-Sheet-Pump Quantization for Hexahedral Meshing." *Computer Graphics Forum*, 43, 5.

H. Brückler and M. Campen. 2026. "Volume Quantization with Flexible Singularities for Hexahedral Meshing." *Computer Graphics Forum*, 45, 2.

H. Brückler, D. Bommes, and M. Campen. 2022a. "Volume Parametrization Quantization for Hexahedral Meshing." *ACM Transactions on Graphics*, 41, 4.

H. Brückler and M. Campen. 2023. "Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing." *ACM Transactions on Graphics*, 42, 6.

H. Brückler, O. Gupta, M. Mandad, and M. Campen. 2022b. "The 3D Motorcycle Complex for Structured Volume Decomposition." *Computer Graphics Forum*, 41, 2.

H. Bruggesser and P. Mani. 1971. "Shellable decompositions of cells and spheres." *Mathematica Scandinavica*, 29, 2, 197–205.

M. Campen. 2017. "Partitioning Surfaces Into Quadrilateral Patches: A Survey." *Computer Graphics Forum*, 36, 8, 567–588.

M. Campen, D. Bommes, and L. Kobbelt. 2015. "Quantized global parametrization." *ACM Transactions on Graphics*, 34, 6.

M. Campen, R. Capouellez, H. Shen, L. Zhu, D. Panozzo, and D. Zorin. 2021. "Efficient and robust discrete conformal equivalence with boundary." *ACM Transactions on Graphics*, 40, 6, Article 261, 16 pages.

M. Campen and L. Kobbelt. 2014. "Quad layout embedding via aligned parameterization." *Computer Graphics Forum*, 33, 8, 69–81.

M. Campen, H. Shen, J. Zhou, and D. Zorin. 2019. "Seamless Parametrization with Arbitrary Cones for Arbitrary Genus." *ACM Transactions on Graphics*, 39, 1.

M. Campen, C. T. Silva, and D. Zorin. 2016. "Bijective maps from simplicial foliations." *ACM Transactions on Graphics*, 35, 4, 1–15.

M. Campen and D. Zorin. 2017. "Similarity maps and field-guided T-splines: a perfect couple." *ACM Transactions on Graphics*, 36, 4, 1–16.

C. Canuto, A. Quarteroni, M. Y. Hussaini, and T. A. Zang. 2007. "Discretization strategies for spectral methods in complex domains." *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*, 237–357.

A. Chemin, F. Henrotte, J.-F. Remacle, and J. V. Schaftingen. 2018. "Representing three-dimensional cross fields using fourth order tensors." In: *International Meshing Roundtable*. Springer, 89–108.

L. Chen, G. Xu, S. Wang, Z. Shi, and J. Huang. 2019. "Constructing volumetric parameterization based on directed graph simplification of $\ell 1$ polycube structure from complex shapes." *Computer Methods in Applied Mechanics and Engineering*, 351, 422–440.

L. Chen and J.-C. Xu. 2004. "Optimal delaunay triangulations." *Journal of Computational Mathematics*, 299–308.

Z. Chen, W. Wang, B. Lévy, L. Liu, and F. Sun. 2014. "Revisiting optimal Delaunay triangulation for 3D graded mesh generation." *SIAM Journal on Scientific Computing*, 36, 3, A930–A954.

S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. 2000. "Sliver exudation." *Journal of the ACM (JACM)*, 47, 5, 883–904.

S.-W. Cheng, T. K. Dey, J. Shewchuk, and S. Sahni. 2013. *Delaunay mesh generation*. CRC Press Boca Raton.

G. Cherchi, P. Alliez, R. Scateni, M. Lyon, and D. Bommes. 2019. "Selective padding for polycube-based hexahedral meshing." *Computer Graphics Forum*, 38, 1, 580–591.

G. Cherchi, M. Livesu, and R. Scateni. 2016. "Polycube simplification for coarse layouts of surfaces and volumes." *Computer Graphics Forum*, 35, 5, 11–20.

L. P. Chew. 1987. "Constrained delaunay triangulations." In: *Proceedings of the third annual symposium on Computational geometry*, 215–222.

P. G. Ciarlet. 2002. *The finite element method for elliptic problems*. SIAM.

D. Cohen-Steiner, E. C. De Verdiere, and M. Yvinec. 2002. "Conforming Delaunay triangulations in 3D." In: *Proceedings of the eighteenth annual symposium on Computational geometry*, 199–208.

G. Coiffier and E. Corman. 2023. "The Method of Moving Frames for Surface Global Parametrization." *ACM Transactions on Graphics*, 42, 5, 1–18.

COMSOL Inc. 2025. *Comsol Multiphysics*. https://www.comsol.com/comsol-multiphysics. (2025).

Coreform Inc. 2025. *Cubit*. https://coreform.com/products/coreform-cubit/government/. (2025).

E. Corman and K. Crane. 2025. "Rectangular Surface Parameterization." *ACM Transactions on Graphics*, 44, 4, 1–21.

E. Corman and K. Crane. 2019. "Symmetric Moving Frames." *ACM Transactions on Graphics*, 38, 4.

J. A. Cottrell, A. Reali, Y. Bazilevs, and T. J. Hughes. 2006. "Isogeometric analysis of structural vibrations." *Computer methods in applied mechanics and engineering*, 195, 41-43, 5257–5296.

Y. Coudert-Osmont, D. Desobry, M. Heistermann, D. Bommes, N. Ray, and D. Sokolov. 2024. "Quad Mesh Quantization Without a T-Mesh." *Computer Graphics Forum*, 43, 1, e14928.

M. Couplet, A. Chemin, and J.-F. Remacle. 2025. "Size-controlled quadrilateral meshing using integrable odeco fields." *Computer-Aided Design*, 103974.

R. Courant, K. Friedrichs, and H. Lewy. 1967. "On the partial difference equations of mathematical physics." *IBM journal of Research and Development*, 11, 2, 215–234.

J.-C. Cuillière, V. Francois, and J.-M. Drouet. 2012. "Automatic 3D mesh generation of multiple domains for topology optimization methods." In: *Proceedings of the 21st International Meshing Roundtable*. Springer, 243–259.

J. Daniels, C. T. Silva, and E. Cohen. 2009. "Localized quadrilateral coarsening." In: *Computer Graphics Forum* 5. Vol. 28. Wiley Online Library, 1437–1444.

Dassault Systèmes Inc. 2025. *Abaqus FEA*. https://www.3ds.com/products/simulia/abaqus. (2025).

D. Desobry, Y. Coudert-Osmont, E. Corman, N. Ray, and D. Sokolov. 2021. "Designing 2D and 3D non-orthogonal frame fields." *Computer-Aided Design*, 139, 103081.

T. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev. 1999. "Topology preserving edge contraction." *Publications de l'Institut Mathématique*, 66.

O. Diamanti, A. Vaxman, D. Panozzo, and O. Sorkine-Hornung. 2015. "Integrable polyvector fields." *ACM Transactions on Graphics*, 34, 4, 1–12.

L. Diazzi, D. Panozzo, A. Vaxman, and M. Attene. 2023. "Constrained delaunay tetrahedrization: A robust and practical approach." *ACM Transactions on Graphics*, 42, 6, 1–15.

C. Doran, A. Chang, and R. Bridson. 2013. "Isosurface stuffing improved: acute lattices and feature matching." In: *ACM SIGGRAPH 2013 Talks*, 1–1.

X. Du, N. Aigerman, Q. Zhou, S. Z. Kovalsky, Y. Yan, D. M. Kaufman, and T. Ju. 2020. "Lifting Simplices to Find Injectivity." *ACM Transactions on Graphics*, 39, 4.

A. Dumitrescu, G. Rote, and C. D. Tóth. 2013. "Monotone paths in planar convex subdivisions and polytopes." In: *Discrete Geometry and Optimization*. Springer, 79–104.

H.-C. Ebke, D. Bommes, M. Campen, and L. Kobbelt. 2013. "QEx: robust quad mesh extraction." *ACM Transactions on Graphics*, 32, 6, 1–10.

M. Eck and H. Hoppe. 1996. "Automatic reconstruction of B-spline surfaces of arbitrary topological type." In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 325–334.

D. Eppstein. 2003. "Dynamic generators of topologically embedded graphs." SODA '03, 599–608.

D. Eppstein and J. Erickson. 1999. "Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions." *Discrete & Computational Geometry*, 22, 4, 569–592.

D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. 2008. "Motorcycle graphs: canonical quad mesh partitioning." *Comp. Graph. Forum*, 27, 5, 1477–1486.

D. Eppstein and E. Mumford. 2010. "Steinitz theorems for orthogonal polyhedra." In: *Proceedings of the twenty-sixth annual symposium on Computational geometry*, 429–438.

J. Erickson. 2013. "Theoretical advances in hexahedral mesh generation." In: *Proc. 29th Annu. Symp. Comput. Geometry Workshop Mesh Gener*, 13–17.

V. Esslinger, R. Kieselbach, R. Koller, and B. Weisse. 2004. "The railway accident of Eschede–technical background." *Engineering Failure Analysis*, 11, 4, 515–535.

X. Fang, J. Huang, Y. Tong, and H. Bao. 2021. "Metric-driven 3D frame field generation." *IEEE Transactions on Visualization and Computer Graphics*, 29, 4, 1964–1976.

X. Fang, W. Xu, H. Bao, and J. Huang. 2016. "All-hex meshing using closed-form induced polycube." *ACM Transactions on Graphics*, 35, 4, 1–9.

L. Feng, P. Alliez, L. Busé, H. Delingette, and M. Desbrun. 2018. "Curved optimal delaunay triangulation." *ACM Transactions on Graphics*, 37, 4, 16.

M. Fisher, B. Springborn, P. Schröder, and A. I. Bobenko. 2007. "An algorithm for the construction of intrinsic Delaunay triangulations with applications to digital geometry processing." *Computing*, 81, 2, 199–213.

N. T. Folwell and S. A. Mitchell. 1999. "Reliable whisker weaving via curve contraction." *Engineering with computers*, 15, 3, 292–302.

L. A. Freitag and C. Ollivier-Gooch. 1997. "Tetrahedral mesh improvement using swapping and smoothing." *International Journal for Numerical Methods in Engineering*, 40, 21, 3979–4002.

X.-M. Fu, C.-Y. Bai, and Y. Liu. 2016. "Efficient volumetric polycube-map construction." *Computer graphics forum*, 35, 7, 97–106.

Y. Funabashi and K. Kitazawa. 2012. "Fukushima in review: A complex disaster, a disastrous response." *Bulletin of the Atomic Scientists*, 68, 2, 9–21.

R. Gao and J. Peters. 2021. "Improving hexahedral-FEM-based plasticity in surgery simulation." In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 571–580.

X. Gao and G. Chen. 2016. "A local frame based hexahedral mesh optimization." *Proceedings of the 25th International Meshing Roundtable*, 2.

X. Gao, Z. Deng, and G. Chen. 2015a. "Hexahedral mesh re-parameterization from aligned base-complex." *ACM Transactions on Graphics*, 34, 4, 1–10.

X. Gao, W. Jakob, M. Tarini, and D. Panozzo. 2017a. "Robust hex-dominant mesh generation using field-guided polyhedral agglomeration." *ACM Transactions on Graphics (TOG)*, 36, 4, 1–13.

X. Gao, T. Martin, S. Deng, E. Cohen, Z. Deng, and G. Chen. 2015b. "Structured volume decomposition via generalized sweeping." *IEEE transactions on visualization and computer graphics*, 22, 7, 1899–1911.

X. Gao, D. Panozzo, W. Wang, Z. Deng, and G. Chen. 2017b. "Robust Structure Simplification for Hex Re-Meshing." *ACM Transactions on Graphics*, 36, 6.

X. Gao, H. Shen, and D. Panozzo. 2019. "Feature preserving octree-based hexahedral meshing." *Computer graphics forum*, 38, 5, 135–149.

V. Garanzha, I. Kaporin, L. Kudryavtseva, F. Protais, N. Ray, and D. Sokolov. 2021. "Foldover-free maps in 50 lines of code." *ACM Transactions on Graphics*, 40, 4, Article 102, 16 pages.

C. Geuzaine and J.-F. Remacle. 2009. "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities." *International journal for numerical methods in engineering*, 79, 11, 1309–1331.

M. Gillespie, N. Sharp, and K. Crane. 2021. "Integer coordinates for intrinsic geometry processing." *ACM Transactions on Graphics*, 40, 6, Article 252, 13 pages.

D. Golovaty, J. A. Montero, and D. Spirn. 2021. "A variational method for generating n-cross fields using higher-order Q-tensors." *SIAM Journal on Scientific Computing*, 43, 5, A3269–A3304.

L. Grady. 2008. "Minimal surfaces extend shortest path segmentation methods to 3D." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32, 2, 321–334.

A. Gravouil, T. Elguedj, and H. Maigre. 2009. "An explicit dynamics extended finite element method. Part 2: Element-by-element stable-explicit/explicit dynamic scheme." *Computer Methods in Applied Mechanics and Engineering*, 198, 30-32, 2318–2328.

J. Gregson, A. Sheffer, and E. Zhang. 2011. "All-hex mesh generation via volumetric polycube deformation." *Computer graphics forum*, 30, 5, 1407–1416.

E. Gunpinar, M. Moriguchi, H. Suzuki, and Y. Ohtake. 2014. "Motorcycle graph enumeration from quadrilateral meshes for reverse engineering." *Computer-Aided Design*, 55, 64–80.

Gurobi Optimization, LLC. 2025. *Gurobi Optimizer*. https://www.gurobi.com. (2025).

R. Haimes. 2015. "MOSS: multiple orthogonal strand system." *Engineering with Computers*, 31, 3, 453–463.

A. Hatcher. 2002. *Algebraic topology*. Cambridge Univ. Press, Cambridge.

S. Hinderink, H. Brückler, and M. Campen. 2024. "Bijective Volumetric Mapping via Star Decomposition." *ACM Transactions on Graphics*, 43, 6, Article 168, 11 pages.

S. Hinderink and M. Campen. 2023. "Galaxy Maps: Localized Foliations for Bijective Volumetric Mapping." *ACM Transactions on Graphics*, 42, 4.

A. J. Hoffman and J. B. Kruskal. 2009. "Integral boundary points of convex polyhedra." In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, 49–76.

K. Hu, J. Qian, and Y. Zhang. 2013. "Adaptive all-hexahedral mesh generation based on a hybrid octree and bubble packing." *22nd International Meshing Roundtable*.

Y. Hu, T. Schneider, B. Wang, D. Zorin, and D. Panozzo. 2020. "Fast tetrahedral meshing in the wild." *ACM Transactions on Graphics (ToG)*, 39, 4, 117–1.

Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo. 2018. "Tetrahedral meshing in the wild." *ACM Transactions on Graphics*, 37, 4, 60.

J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. 2014a. "$\ell 1$-based construction of polycube maps from complex shapes." *ACM Transactions on Graphics*, 33, 3, 1–11.

J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. 2014b. "$\ell 1$-based construction of polycube maps from complex shapes." *ACM Transactions on Graphics*, 33, 3, 1–11.

J. Huang, Y. Tong, H. Wei, and H. Bao. 2011. "Boundary aligned smooth 3D cross-frame field." *ACM Transactions on Graphics*, 30, 6.

T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. 2005. "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement." *Computer methods in applied mechanics and engineering*, 194, 39-41, 4135–4195.

M. Isenburg and P. Alliez. 2002. "Compressing hexahedral volume meshes." In: *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.* IEEE, 284–293.

C. Jamin, P. Alliez, M. Yvinec, and J.-D. Boissonnat. 2015. "CGALmesh: a generic framework for delaunay mesh generation." *ACM Transactions on Mathematical Software (TOMS)*, 41, 4, 1–24.

T. Jiang, J. Huang, Y. Wang, Y. Tong, and H. Bao. 2014. "Frame field singularity correction for automatic hexahedralization." *IEEE TVCG*, 20, 8, 1189–1199.

F. Kälberer, M. Nieser, and K. Polthier. 2007. "QuadCover - Surface Parameterization using Branched Coverings." *Computer Graphics Forum*, 26, 3, 375–384.

T. Kanai, H. Suzuki, and F. Kimura. 1997. "3D geometric metamorphosis based on harmonic map." In: *Proceedings The Fifth Pacific Conference on Computer Graphics and Applications*, 97–104.

K. Karčiauskas, D. Panozzo, and J. Peters. 2017. "T-junctions in Spline Surfaces." *ACM Transactions on Graphics*, 36, 5, 1–9.

J. Kim, M. Jin, Q.-Y. Zhou, F. Luo, and X. Gu. 2008. "Computing fundamental group of general 3-manifold." In: *International Symposium on Visual Computing*. Springer, 965–974.

B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. 2006. "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations." *Engineering with Computers*, 22, 3, 237–254.

R. Klette. 2000. "Cell complexes through time." In: *Vision Geometry IX*. Ed. by L. J. Latecki, D. M. Mount, and A. Y. Wu. Vol. 4117. International Society for Optics and Photonics. SPIE, 134–145.

C. Knauer and D. Werner. 2012. "Erdos-Szekeres is NP-hard in 3 dimensions—and what now." *28th EuroCG*, 61–64.

P. Knupp. 1995. "Mesh generation using vector fields." *Journal of Computational Physics*, 119, 1, 142–148.

P. M. Knupp. 2003. "A method for hexahedral mesh shape optimization." *International journal for numerical methods in engineering*, 58, 2, 319–332.

P. M. Knupp. 2000. "Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II—a framework for volume mesh optimization and the condition number of the Jacobian matrix." *International Journal for numerical methods in engineering*, 48, 8, 1165–1185.

P. M. Knupp. 1990. "On the invertibility of the isoparametric map." *Computer Methods in Applied Mechanics and Engineering*, 78, 3, 313–329.

T. Kohler, M. Heistermann, and D. Bommes. 2025. "HexHex: Highspeed Extraction of Hexahedral Meshes." *ACM Transactions on Graphics*, 44, 4, Article 147, 20 pages.

D. A. Kopriva. 2006. "Metric identities and the discontinuous spectral element method on curvilinear meshes." *Journal of Scientific Computing*, 26, 3, 301–327.

D. A. Kopriva. 2009. "Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers." Springer Netherlands. Chap. Spectral Element Methods, 293–354.

N. Kowalski, F. Ledoux, M. L. Staten, and S. J. Owen. 2012. "Fun sheet matching: towards automatic block decomposition for hexahedral meshes." *Engineering with Computers*, 28, 3, 241–253.

V. Kraevoy and A. Sheffer. 2004. "Cross-parameterization and compatible remeshing of 3D models." *ACM Transactions on Graphics*, 23, 3, 861–869.

V. Kraevoy, A. Sheffer, and C. Gotsman. 2003. "Matchmaker: constructing constrained texture maps." *ACM Transactions on Graphics (ToG)*, 22, 3, 326–333.

M. Kremer, D. Bommes, and L. Kobbelt. 2013. "OpenVolumeMesh–A versatile index-based data structure for 3D polytopal complexes." In: *Proceedings of the 21st International Meshing Roundtable*, 531–548.

M. Kremer, D. Bommes, I. Lim, and L. Kobbelt. 2014. "Advanced automatic hexahedral mesh generation from surface quad meshes." In: *Proceedings of the 22nd International Meshing Roundtable*. Springer, 147–164.

F. Labelle and J. R. Shewchuk. 2007. "Isosurface stuffing: fast tetrahedral meshes with good dihedral angles." In: *ACM SIGGRAPH 2007 papers*, 57–es.

F. Ledoux and J. Shepherd. 2010. "Topological modifications of hexahedral meshes via sheet operations: a theoretical study." *Eng. with Comput.*, 26, 4, 433–447.

F. Ledoux and J.-C. Weill. 2008. "An extension of the reliable whisker weaving algorithm." In: *Proceedings of the 16th international meshing roundtable*. Springer, 215–232.

D.-T. Lee and A. K. Lin. 1986. "Generalized Delaunay triangulation for planar graphs." *Discrete & Computational Geometry*, 1, 3, 201–217.

Z. Levi. 2022. "Seamless parametrization of spheres with controlled singularities." 41, 1, 57–68.

Z. Levi. 2023. "Seamless Parametrization with Cone and Partial Loop Control." *ACM Transactions on Graphics*, 42, 5, 1–22.

L. Li, P. Zhang, D. Smirnov, S. M. Abulnaga, and J. Solomon. 2021. "Interactive all-hex meshing via cuboid decomposition." *ACM Transactions on Graphics*, 40, 6, 1–17.

T. Li, R. McKeag, and C. Armstrong. 1995. "Hexahedral meshing using midpoint subdivision and integer programming." *Computer methods in applied mechanics and engineering*, 124, 1-2, 171–193.

Y. Li, Y. Liu, W. Xu, W. Wang, and B. Guo. 2012. "All-hex meshing using singularity-restricted field." *ACM Transactions on Graphics*, 31, 6, 177.

H. Lin, S. Jin, H. Liao, and Q. Jian. 2015. "Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm." *Computer-Aided Design*, 67, 107–117.

J. Lin, X. Jin, Z. Fan, and C. C. Wang. 2008. "Automatic polycube-maps." In: *International Conference on Geometric Modeling and Processing*. Springer, 3–16.

H.-Y. Liu, Z.-Y. Liu, Z.-Y. Zhao, L. Liu, and X.-M. Fu. 2020. "Practical Fabrication of Discrete Chebyshev Nets." *Comp. Graph. Forum*, 39, 7, 13–26.

H. Liu and D. Bommes. 2023. "Locally Meshable Frame Fields." *ACM Transactions on Graphics*, 42, 4, Article 112, 20 pages.

H. Liu, P. Zhang, E. Chien, J. Solomon, and D. Bommes. 2018. "Singularity-constrained octahedral fields for hexahedral meshing." *ACM Transactions on Graphics*, 37, 4.

S.-S. Liu, J. Uicker Jr, and R. Gadh. 1999. "A dual geometry—topology constraint approach for determination of pseudo-swept shapes as applied to hexahedral mesh generation." *Computer-aided design*, 31, 6, 413–426.

S.-S. Liu and R. Gadh. 1997. "Automatic hexahedral mesh generation by recursive convex and swept volume decomposition." In: *6th international meshing roundtable, Sandia National Laboratories*. Citeseer, 217–231.

M. Livesu, M. Attene, G. Patané, and M. Spagnuolo. 2017. "Explicit cylindrical maps for general tubular shapes." *Computer-Aided Design*, 90, 27–36.

M. Livesu, A. Muntoni, E. Puppo, and R. Scateni. 2016. "Skeleton-driven adaptive hexahedral meshing of tubular shapes." *Computer graphics forum*, 35, 7, 237–246.

M. Livesu, N. Pietroni, E. Puppo, A. Sheffer, and P. Cignoni. 2020. "LoopyCuts: practical feature-preserving block decomposition for strongly hex-dominant meshing." *ACM Transactions on Graphics*, 39, 4, 121.

M. Livesu, L. Pitzalis, and G. Cherchi. 2021. "Optimal dual schemes for adaptive grid based hexmeshing." *ACM Transactions on Graphics*, 41, 2, 1–14.

M. Livesu, A. Sheffer, N. Vining, and M. Tarini. 2015. "Practical hex-mesh optimization via edge-cone rectification." *ACM Transactions on Graphics*, 34, 4, 1–11.

M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni. 2013. "Polycut: Monotone graph-cuts for polycube base-complex construction." *ACM Transactions on Graphics*, 32, 6, 1–12.

R. Lougee-Heimer. 2003. "The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community." *IBM Journal of Research and Development*, 47, 1, 57–66.

J. H.-C. Lu, W. R. Quadros, and K. Shimada. 2017. "Evaluation of user-guided semi-automatic decomposition tool for hexahedral mesh generation." *Journal of Computational Design and Engineering*, 4, 4, 330–338.

M. Lyon, D. Bommes, and L. Kobbelt. 2016. "HexEx: Robust Hexahedral Mesh Extraction." *ACM Transactions on Graphics*, 35, 4.

M. Lyon, M. Campen, D. Bommes, and L. Kobbelt. 2019. "Parametrization Quantization with Free Boundaries for Trimmed Quad Meshing." *ACM Transactions on Graphics*, 38, 4.

M. Lyon, M. Campen, and L. Kobbelt. 2021a. "Quad Layouts via Constrained T-Mesh Quantization." *Computer Graphics Forum*, 40, 2.

M. Lyon, M. Campen, and L. Kobbelt. 2021b. "Simpler Quad Layouts using Relaxed Singularities." *Computer Graphics Forum*, 40, 5, 169–179.

M. Mandad and M. Campen. 2019. "Exact constraint satisfaction for truly seamless parametrization." *Computer Graphics Forum*, 38, 2, 135–145.

M. Mandad, R. Chen, D. Bommes, and M. Campen. 2022. "Intrinsic mixed-integer polycubes for hexahedral meshing." *Computer aided geometric design*, 94, 102078.

M. Mandad, D. Cohen-Steiner, and P. Alliez. 2015. "Isotopic approximation within a tolerance volume." *ACM Transactions on Graphics*, 34, 4, 1–12.

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

Z. Marschner, D. Palmer, P. Zhang, and J. Solomon. 2020. "Hexahedral Mesh Repair via Sum-of-Squares Relaxation." In: *Computer Graphics Forum* 5. Vol. 39. Wiley Online Library, 133–147.

F. Meloni, G. Cherchi, R. Scateni, and M. Livesu. 2024. "To What Extent Are Existing Volume Mapping Algorithms Practically Useful?" In: *Smart Tools and Applications in Graphics - Eurographics Italian Chapter Conference*. Ed. by A. Caputo, V. Garro, A. Giachetti, U. Castellani, and T. G. Dulecha. The Eurographics Association. ISBN: 978-3-03868-265-3.

A. de Mesmay, Y. Rieck, E. Sedgwick, and M. Tancer. 2018. "Embeddability in $\mathbb{R}^3$ is NP-hard." In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1316–1329.

A. Mitchell, G. Phillips, and E. Wachspress. 1971. "Forbidden shapes in the finite element method." *IMA Journal of Applied Mathematics*, 8, 2, 260–269.

S. A. Mitchell. 1996. "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, 465–476.

S. A. Mitchell. 2000. "High fidelity interval assignment." *Int. Journal of Computational Geometry & Applications*, 10, 04, 399–415.

S. A. Mitchell. 2023. "Incremental Interval Assignment by Integer Linear Algebra with Improvements." *Computer-Aided Design*, 158, 103485.

S. A. Mitchell. 2014. "Simple and Fast Interval Assignment Using Nonlinear and Piecewise Linear Objectives." In: *Proceedings of the 22nd International Meshing Roundtable*. Ed. by J. Sarrate and M. Staten. Springer Cham, 203–221. ISBN: 978-3-319-02335-9.

A. Moraes, P. Lage, G. Cunha, and L. da Silva. 2013. "Analysis of the non-orthogonality correction of finite volume discretization on unstructured meshes." In: *Proceedings of the 22nd international congress of mechanical engineering, Ribeirão Preto, Brazil*, 3–7.

M. Müller-Hannemann. 2004. "Shelling hexahedral complexes for mesh generation." In: *Graph Algorithms And Applications 2*. World Scientific, 475–507.

M. Müller-Hannemann and K. Weihe. 1997. "Minimum strictly convex quadrangulations of convex polygons." In: *Proceedings of the thirteenth annual symposium on Computational geometry*, 193–202.

M. Murphy, D. M. Mount, and C. W. Gable. 2001. "A point-placement strategy for conforming Delaunay tetrahedralization." *International Journal of Computational Geometry & Applications*, 11, 06, 669–682.

O. R. Musin. 1997. "Properties of the Delaunay triangulation." In: *Proceedings of the thirteenth annual symposium on Computational geometry*, 424–426.

A. Myles, N. Pietroni, D. Kovacs, and D. Zorin. 2010. "Feature-aligned T-meshes." *ACM Transactions on Graphics*, 29, 4, 1–11.

A. Myles, N. Pietroni, and D. Zorin. 2014. "Robust field-aligned global parametrization." *ACM Transactions on Graphics*, 33, 4, Article 135, 14 pages.

A. Myles and D. Zorin. 2012. "Global parametrization by incremental flattening." *ACM Transactions on Graphics*, 31, 4, 109:1–109:11.

M. Nieser, U. Reitebuch, and K. Polthier. 2011. "CubeCover – Parameterization of 3D Volumes." *Computer Graphics Forum*, 30, 5, 1397–1406.

V. Z. Nigolian, M. Campen, and D. Bommes. 2023. "Expansion Cones: A Progressive Volumetric Mapping Framework." *ACM Transactions on Graphics*, 42, 4.

V. Z. Nigolian, M. Campen, and D. Bommes. 2024. "A Progressive Embedding Approach to Bijective Tetrahedral Maps driven by Cluster Mesh Topology." *ACM Transactions on Graphics*, 43, 6, Article 170, 14 pages.

S. Nuvoli, A. Hernandez, C. Esperança, R. Scateni, P. Cignoni, and N. Pietroni. 2019. "QuadMixer: layout preserving blending of quadrilateral meshes." *ACM Transactions on Graphics*, 38, 6, 1–13.

S. J. Owen and S. Saigal. 2000. "H-Morph: an indirect approach to advancing front hex meshing." *International Journal for Numerical Methods in Engineering*, 49, 1-2, 289–312.

G.-P. Paillé, N. Ray, P. Poulin, A. Sheffer, and B. Lévy. 2015. "Dihedral angle-based maps of tetrahedral meshes." *ACM Transactions on Graphics*, 34, 4, 1–10.

D. Palmer, D. Bommes, and J. Solomon. 2020. "Algebraic Representations for Volumetric Frame Fields." *ACM Transactions on Graphics*, 39, 2.

C. H. Papadimitriou and K. Steiglitz. 1998. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

C.-H. Peng, E. Zhang, Y. Kobayashi, and P. Wonka. 2011. "Connectivity editing for quadrilateral meshes." In: *Proceedings of the 2011 SIGGRAPH Asia conference*, 1–12.

N. Pietroni, S. Nuvoli, T. Alderighi, P. Cignoni, and M. Tarini. 2021. "Reliable feature-line driven quad-remeshing." *ACM Transactions on Graphics*, 40, 4, Article 155, 17 pages.

N. Pietroni, E. Puppo, G. Marcias, R. Scopigno, and P. Cignoni. 2016. "Tracing Field-Coherent Quad Layouts." *Computer Graphics Forum*, 35, 7, 485–496.

N. Pietroni et al.. 2022. "Hex-Mesh Generation and Processing: A Survey." *ACM Transactions on Graphics*, 42, 2, Article 16, 44 pages.

L. Pitzalis, M. Livesu, G. Cherchi, E. Gobbetti, and R. Scateni. 2021. "Generalized adaptive refinement for grid-based hexahedral meshing." *ACM Transactions on Graphics*, 40, 6, 1–13.

J. Pommier and Y. Renard. 2005. *Getfem++, an open source generic C++ library for finite element methods*. (2005).

E. Praun, W. Sweldens, and P. Schröder. 2001. "Consistent mesh parameterizations." In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 179–184.

M. A. Price and C. G. Armstrong. 1997. "Hexahedral mesh generation by medial surface subdivision: Part II. Solids with flat and concave edges." *International Journal for Numerical Methods in Engineering*, 40, 1, 111–136.

F. Protais, M. Reberol, N. Ray, E. Corman, F. Ledoux, and D. Sokolov. 2022. "Robust quantization for polycube maps." (2022).

M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung. 2017. "Scalable Locally Injective Mappings." *ACM Transactions on Graphics*, 36, 4.

N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez. 2006. "Periodic global parameterization." *ACM Transactions on Graphics*, 25, 4, 1460–1485.

N. Ray, D. Sokolov, and B. Lévy. 2016. "Practical 3D frame field generation." *ACM Transactions on Graphics*, 35, 6, 1–9.

N. Ray, B. Vallet, W. C. Li, and B. Lévy. 2008. "N-symmetry direction field design." *ACM Transactions on Graphics*, 27, 2, 1–13.

F. Razafindrazaka and K. Polthier. 2017. "Optimal base complexes for quadrilateral meshes." *Computer Aided Geometric Design*, 52, 63–74.

M. Reberol, A. Chemin, and J.-F. Remacle. 2019. "Multiple Approaches to Frame Field Correction for CAD Models." In: *Proc. 28th International Meshing Roundtable*, 283–295.

E. Ruiz-Gironés, X. Roca, and J. Sarrate. 2014. "Optimizing mesh distortion by hierarchical iteration relocation of the nodes on the CAD entities." *Procedia Engineering*, 82, 101–113.

E. Ruiz-Gironés, X. Roca, and J. Sarrate. 2012. "The receding front method applied to hexahedral mesh generation of exterior domains." *Engineering with computers*, 28, 4, 391–408.

E. Ruiz-Gironés, X. Roca, J. Sarrate, R. Montenegro, and J. M. Escobar. 2015. "Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework." *Advances in Engineering Software*, 80, 12–24.

J. Ruppert and R. Seidel. 1992. "On the difficulty of triangulating three-dimensional nonconvex polyhedra." *Discrete & Computational Geometry*, 7, 3, 227–253.

R. R. Salagame and A. D. Belegundu. 1994. "Distortion, degeneracy and rezoning in finite elements—a survey." *Sadhana*, 19, 2, 311–335.

J. Sarrate Ramos, E. Ruiz-Gironés, and F. J. Roca Navarro. 2014. "Unstructured and semi-structured hexahedral mesh generation methods." *Computational Technology Reviews*, 10, 35–64.

N. Schertler, D. Panozzo, S. Gumhold, and M. Tarini. 2018. "Generalized motorcycle graphs for imperfect quad-dominant meshes." *ACM Transactions on Graphics*, 37, 4.

P. Schmidt, J. Born, D. Bommes, M. Campen, and L. Kobbelt. 2022. "TinyAD: Automatic differentiation in geometry processing made simple." In: *Computer graphics forum* 5. Vol. 41. Wiley Online Library, 113–124.

P. Schmidt, J. Born, M. Campen, and L. Kobbelt. 2019. "Distortion-minimizing injective maps between surfaces." *ACM Transactions on Graphics*, 38, 6.

P. Schmidt, M. Campen, J. Born, and L. Kobbelt. 2020. "Inter-surface maps via constant-curvature metrics." *ACM Transactions on Graphics*, 39, 4, 119–1.

T. Schneider, J. Dumas, X. Gao, D. Zorin, and D. Panozzo. 2019. "PolyFEM."

T. Schneider, Y. Hu, X. Gao, J. Dumas, D. Zorin, and D. Panozzo. 2022. "A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method." *ACM Transactions on Graphics*, 41, 3, Article 23, 14 pages.

R. Schneiders. 1996. "A grid-based algorithm for the generation of hexahedral element meshes." *Engineering with computers*, 12, 3, 168–177.

J. Schöberl. 1997. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules." *Computing and visualization in science*, 1, 1, 41–52.

E. Schönhardt. 1928. "Über die zerlegung von dreieckspolyedern in tetraeder." *Mathematische Annalen*, 98, 1, 309–312.

J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. 2004. "Inter-surface mapping." In: *ACM SIGGRAPH 2004 Papers*, 870–877.

M. A. Scott, X. Li, T. W. Sederberg, and T. J. Hughes. 2012. "Local refinement of analysis-suitable T-splines." *Comp. Meth. Appl. Mech. Eng.*, 213.

M. Senechal. 1981. "Which tetrahedra fill space?" *Mathematics magazine*, 54, 5, 227–243.

F. Shang, Y. Gan, and Y. Guo. 2017. "Hexahedral mesh generation via constrained quadrilateralization." *PloS one*, 12, 5.

N. Sharp, Y. Soliman, and K. Crane. 2019. "Navigating intrinsic triangulations." *ACM Transactions on Graphics*, 38, 4, 1–16.

A. Sheffer, M. Etzion, A. Rappoport, and M. Bercovier. 1999. "Hexahedral mesh generation using the embedded Voronoi graph." *Eng. with Comput.*, 15, 3.

C. Shen, J. F. O'Brien, and J. R. Shewchuk. 2004. "Interpolating and approximating implicit surfaces from polygon soup." In: *ACM SIGGRAPH 2004 Papers*, 896–904.

C. Shen, S. Gao, and R. Wang. 2021. "Topological operations for editing the singularity on a hex mesh." *Engineering with Computers*, 37, 2, 1357–1375.

H. Shen, L. Zhu, R. Capouellez, D. Panozzo, M. Campen, and D. Zorin. 2022. "Which cross fields can be quadrangulated? Global parameterization from prescribed holonomy signatures." *ACM Transactions on Graphics*, 41, 4, 1–12.

J. F. Shepherd and C. R. Johnson. 2008. "Hexahedral mesh generation constraints." *Engineering with Computers*, 24, 3, 195–213.

J. R. Shewchuk. 2002a. "Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery." *IMR*, 193, 204.

J. R. Shewchuk. 2002b. "What is a good linear element? Interpolation, conditioning, and quality measures." In: *11th International Meshing Roundtable,{IMR} 2002*.

J. R. Shewchuk and H. Si. 2014. "Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement." In: *Proceedings of the thirtieth annual symposium on Computational geometry*, 290–299.

B. Shin. 1996. "Automated hexahedral mesh generation by swept volume decomposition and recomposition." *5th International Meshing Roundtable, 1996*.

H. Si. 2008. "Adaptive tetrahedral mesh generation by constrained Delaunay refinement." *International Journal for Numerical Methods in Engineering*, 75, 7, 856–880.

H. Si. 2015. "TetGen, a Delaunay-based quality tetrahedral mesh generator." *ACM Trans. Math. Softw*, 41, 2, 11.

H. Si and K. Gärtner. 2005. "Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations." In: *Proceedings of the 14th international meshing roundtable*. Springer, 147–163.

J. Smith and S. Schaefer. 2015. "Bijective parameterization with free boundaries." *ACM Transactions on Graphics*, 34, 4, Article 70, 9 pages.

M. Snoep, B. Speckmann, and K. Verbeek. 2025a. "Polycubes via Dual Loops." In: *Proceedings of the 2025 SIAM International Meshing Roundtable*. SIAM, 72–84.

M. Snoep, B. Speckmann, and K. Verbeek. 2025b. "Robust Construction of Polycube Segmentations via Dual Loops." *Computer Graphics Forum*, 44, 5, e70195.

D. Sokolov. 2016. "Modélisation géométrique." PhD thesis. Université de Lorraine (UL), Vandoeuvre-lès-Nancy, FRA.

D. Sokolov and N. Ray. 2015. *Fixing normal constraints for generation of polycubes*. Research Report. LORIA.

J. Solomon, A. Vaxman, and D. Bommes. 2017. "Boundary element octahedral fields in volumes." *ACM Transactions on Graphics*, 36, 4, 1.

T. Sorgente, S. Biasotti, G. Manzini, and M. Spagnuolo. 2023. "A survey of indicators for mesh quality assessment." *Computer graphics forum*, 42, 2, 461–483.

M. L. Staten, R. A. Kerr, S. J. Owen, T. D. Blacker, M. Stupazzini, and K. Shimada. 2010. "Unconstrained plastering—Hexahedral mesh generation via advancing-front geometry decomposition." *International journal for numerical methods in engineering*, 81, 2, 135–171.

C. Stimpson, C. Ernst, D. C. Thompson, P. M. Knupp, and P. P. Pébay. 2007. *The verdict geometric quality library*. 1751. Sandia National Laboratories.

S. C. Tadepalli, A. Erdemir, and P. R. Cavanagh. 2011. "Comparison of hexahedral and tetrahedral elements in finite element analysis of the foot and footwear." *Journal of biomechanics*, 44, 12, 2337–2343.

K. Takayama. 2019. "Dual Sheet Meshing: An Interactive Approach to Robust Hexahedralization." *Computer Graphics Forum*, 38, 2, 37–48.

T. Tam and C. G. Armstrong. 1993. "Finite element mesh control by integer programming." *International journal for numerical methods in engineering*, 36, 15, 2581–2605.

M. Tarini, K. Hormann, P. Cignoni, and C. Montani. 2004. "Polycube-maps." *ACM transactions on graphics*, 23, 3, 853–860.

T. J. Tautges. 2004. "MOAB-SD: Integrated structured and unstructured mesh representation." *Engineering With Computers*, 20, 3, 286–293.

T. J. Tautges. 2001. "The generation of hexahedral meshes for assembly geometry: survey and progress." *International Journal for Numerical Methods in Engineering*, 50, 12, 2617–2642.

T. J. Tautges, T. Blacker, and S. A. Mitchell. 1996. "The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes." *International Journal for Numerical Methods in Engineering*, 39, 19, 3327–3349.

T. J. Tautges and S. E. Knoop. 2003. "Topology modification of hexahedral meshes using atomic dual-based operations." In: *Proc. 12th International Meshing Roundtable*, 415–423.

C. Thieulot and W. Bangerth. 2025. "On the choice of finite element for applications in geodynamics–Part 2: A comparison of simplex and hypercube elements." *Solid Earth*, 16, 6, 457–476.

Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. 2006a. "Designing Quadrangulations with Discrete Harmonic Forms." In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (SGP '06). Eurographics Association, 201–210. ISBN: 3905673363.

Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. 2006b. "Designing quadrangulations with discrete harmonic forms." In: *Eurographics symposium on geometry processing*.

J. Tournois, C. Wormser, P. Alliez, and M. Desbrun. 2009. "Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation." *ACM Transactions on Graphics*, 28, 3, Article 75, 9 pages.

I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. 1998. *Simplification of tetrahedral meshes*. IEEE.

P. M. Tuchinsky and B. W. Clark. 1997. "The "HexTet" hex-dominant automesher: An interim progress report." In: *Proceedings of the 6th International Meshing Roundtable, Park City, Utah, Sandia National Laboratories, Albuquerque, USA*, 183–193.

W. T. Tutte. 1963. "How to draw a graph." *Proc. Lond. Math. Soc.*, 13, 743–767.

F. Usai, M. Livesu, E. Puppo, M. Tarini, and R. Scateni. 2015. "Extraction of the quad layout of a triangle mesh guided by its curve skeleton." *ACM Transactions on Graphics (TOG)*, 35, 1, 1–13.

D. Vartziotis and M. Papadrakakis. 2017. "Improved GETMe by adaptive mesh smoothing." *Computer Assisted Methods in Engineering and Science*, 20, 1, 55–71.

A. Vaxman, M. Campen, O. Diamanti, D. Panozzo, D. Bommes, K. Hildebrandt, and M. Ben-Chen. 2016. "Directional Field Synthesis, Design, and Processing." *Computer Graphics Forum*, 35, 2.

J. Vekhter, Z. Chen, and E. Vouga. 2025. "Mint: Discretely Integrable Moments for Symmetric Frame Fields." *Computer Graphics Forum*.

K. Verhetsel, J. Pellerin, and J.-F. Remacle. 2019. "A 44-element mesh of Schneiders' pyramid: Bounding the difficulty of hex-meshing problems." *Computer-Aided Design*, 116, 102735.

R. Viertel, M. L. Staten, and F. Ledoux. 2016. *Analysis of Non-Meshable Automatically Generated Frame Fields*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

E. Wang, T. Nelson, and R. Rauch. 2004. "Back to elements-tetrahedra vs. hexahedra." In: *Proceedings of the 2004 international ANSYS conference*.

W. Wang, Y. Zhang, G. Xu, and T. J. R. Hughes. 2012. "Converting an unstructured quadrilateral/hexahedral mesh to a rational T-spline." *Computational Mechanics*, 50, 1, 65–84.

J. Wartman, D. R. Montgomery, S. A. Anderson, J. R. Keaton, J. Benoît, J. dela Chapelle, and R. Gilbert. 2016. "The 22 March 2014 Oso landslide, Washington, USA." *Geomorphology*, 253, 275–288.

D. F. Watson. 1981. "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes." *The computer journal*, 24, 2, 167–172.

I. Weber, G. Kreiss, and M. Nazarov. 2022. "Stability analysis of high order methods for the wave equation." *Journal of Computational and Applied Mathematics*, 404, 113900.

O. Weber and D. Zorin. 2014. "Locally injective parametrization with arbitrary fixed boundaries." *ACM Transactions on Graphics*, 33, 4, 1–12.

M. Weiss, T. Kalscheuer, and Z. Ren. 2023. "Spectral element method for 3-D controlled-source electromagnetic forward modelling using unstructured hexahedral meshes." *Geophysical Journal International*, 232, 2, 1427–1454.

D. Werner. 2013. "Computational aspects of some problems from discrete geometry in higher dimensions." Dissertation.

P. Wesseling and C. W. Oosterlee. 2001. "Geometric multigrid with applications to computational fluid dynamics." *Journal of computational and applied mathematics*, 128, 1-2, 311–334.

D. R. White, S. Saigal, and S. J. Owen. 2004. "CCSweep: automatic decomposition of multi-sweep volumes." *Engineering with computers*, 20, 3, 222–236.

T. J. Wilson, J. Sarrate Ramos, X. Roca Ramón, R. Montenegro, and J. Escobar. 2012. "Untangling and smoothing of quadrilateral and hexahedral meshes." *Civil-Comp Proceedings*.

H. Wu and S. Gao. 2014. "Automatic swept volume decomposition based on sweep directions extraction for hexahedral meshing." *Procedia Engineering*, 82, 136–148.

H. Wu, S. Gao, R. Wang, and J. Chen. 2018. "Fuzzy clustering based pseudo-swept volume decomposition for hexahedral meshing." *Computer-Aided Design*, 96, 42–58.

H. Wu, S. Gao, R. Wang, and M. Ding. 2017. "A global approach to multi-axis swept mesh generation." *Procedia Engineering*, 203, 414–426.

G. Xu, R. Ling, Y. J. Zhang, Z. Xiao, Z. Ji, and T. Rabczuk. 2021. "Singularity structure simplification of hexahedral meshes via weighted ranking." *Computer-Aided Design*, 130, 102946.

K. Xu, X. Gao, and G. Chen. 2018. "Hexahedral mesh quality improvement via edge-angle optimization." *Computers & Graphics*, 70, 17–27.

K. Xu, X. Gao, Z. Deng, and G. Chen. 2017. "Hexahedral meshing with varying element sizes." In: *Computer Graphics Forum* 8. Vol. 36, 540–553.

A. E. Yilmaz and M. Kuzuoglu. 2009. "A particle swarm optimization approach for hexahedral mesh smoothing." *International journal for numerical methods in fluids*, 60, 1, 55–78.

Y. Yu and X. Wei. 2020. "HexGen and Hex2Spline: polycube-based hexahedral mesh generation and unstructured spline construction for isogeometric analysis framework in LS-DYNA." In: *Springer INdAM Serie: Proceedings of INdAM Workshop "Geometric Challenges in Isogeometric Analysis."*

P. Zhang, J. Chiang, X. Fan, and K. Mundilova. 2023. "Local Decomposition of Hexahedral Singular Nodes into Singular Curves." *Computer-Aided Design*, 158, 103484.

P. Zhang, J. Vekhter, E. Chien, D. Bommes, E. Vouga, and J. Solomon. 2020. "Octahedral Frames for Feature-Aligned Cross Fields." *ACM Transactions on Graphics*, 39, 3.

Y. Zhang, Y. Bazilevs, S. Goswami, C. L. Bajaj, and T. J. Hughes. 2007. "Patient-specific vascular NURBS modeling for isogeometric analysis of blood flow." *Computer methods in applied mechanics and engineering*, 196, 29-30, 2943–2959.

Y. Zhang, W. Wang, and T. J. Hughes. 2012. "Solid T-spline construction from boundary representations for genus-zero geometry." *Computer Methods in Applied Mechanics and Engineering*, 249-252, 185–197.

J. Zhou, C. Tu, D. Zorin, and M. Campen. 2020. "Combinatorial construction of seamless parameter domains." *Computer Graphics Forum*, 39, 2, 179–190.

Q. Zhou and A. Jacobson. 2016. "Thingi10k: A dataset of 10,000 3d-printing models." *arXiv preprint arXiv:1605.04797*.

M. Zlámal. 1968. "On the finite element method." *Numerische Mathematik*, 12, 5, 394–409.