Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing

HENDRIK BRÜCKLER and MARCEL CAMPEN, Osnabrück University, Germany

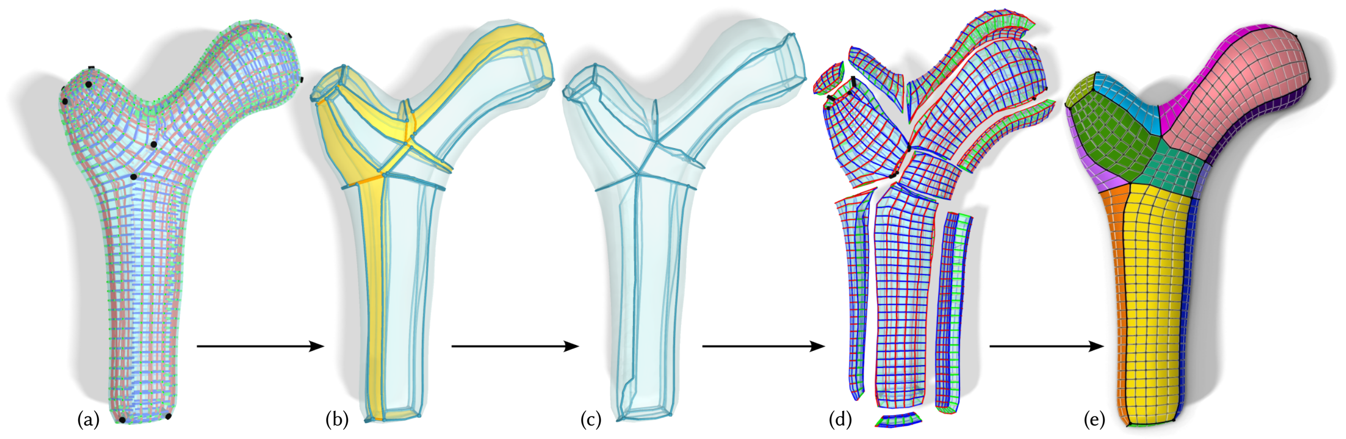


Fig. 1. (a) Visualization of a so-called *seamless parametrization* in the volume of an object. (b) Its *motorcycle complex*, an embedded volumetric cell complex aligned with this parametrization; some of its elements (marked yellow, orange) are determined in a *quantization* process to be undesirable. (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.

We present a set of operators to perform modifications, in particular collapses and splits, in volumetric cell complexes which are discretely embedded in a background mesh. Topological integrity and geometric embedding validity are carefully maintained. We apply these operators strategically to volumetric block decompositions, so-called T-meshes or base complexes, in the context of hexahedral mesh generation. This allows circumventing the expensive and unreliable global volumetric remapping step in the versatile meshing pipeline based on 3D integer-grid maps. In essence, we reduce this step to simpler local cube mapping problems, for which reliable solutions are available. As a consequence, the robustness of the mesh generation process is increased, especially when targeting coarse or block-structured hexahedral meshes. We furthermore extend this pipeline to support feature alignment constraints, and systematically respect these throughout, enabling the generation of meshes that align to points, curves, and surfaces of special interest, whether on the boundary or in the interior of the domain.

CCS Concepts: • **Computing methodologies → Computer graphics**; **Mesh models**; **Mesh geometry models**; *Shape modeling*.

Additional Key Words and Phrases: block-structured, multi-block, T-mesh, hexahedral mesh, volume mesh, block decomposition, base complex

ACM Reference Format:
Hendrik Brückler and Marcel Campen. 2023. Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing. *ACM Trans. Graph.* 42, 6, Article 180 (December 2023), 24 pages. https://doi.org/10.1145/3618384

Authors' address: Hendrik Brückler, hendrik.brueckler@uos.de; Marcel Campen, campen@uos.de, Osnabrück University, Germany.

1 INTRODUCTION

Mapping problems between geometric objects are very challenging when properties like injectivity or bijectivity are required. Reliable algorithmic solutions with strict guarantees 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].

These examples concern the case of 2D maps. We want to follow such a route in 3D, in the context of hexahedral mesh generation using *volumetric integer-grid maps* (IGMs), a state-of-the-art approach to this important task [Pietroni et al. 2022; Liu and Bommes 2023]. These maps are volumetric, their 3D domains are complex (non-convex and even self-overlapping), and local injectivity is strictly required, together making their computation a highly challenging problem without a reliable solution so far. Our goal herein is to

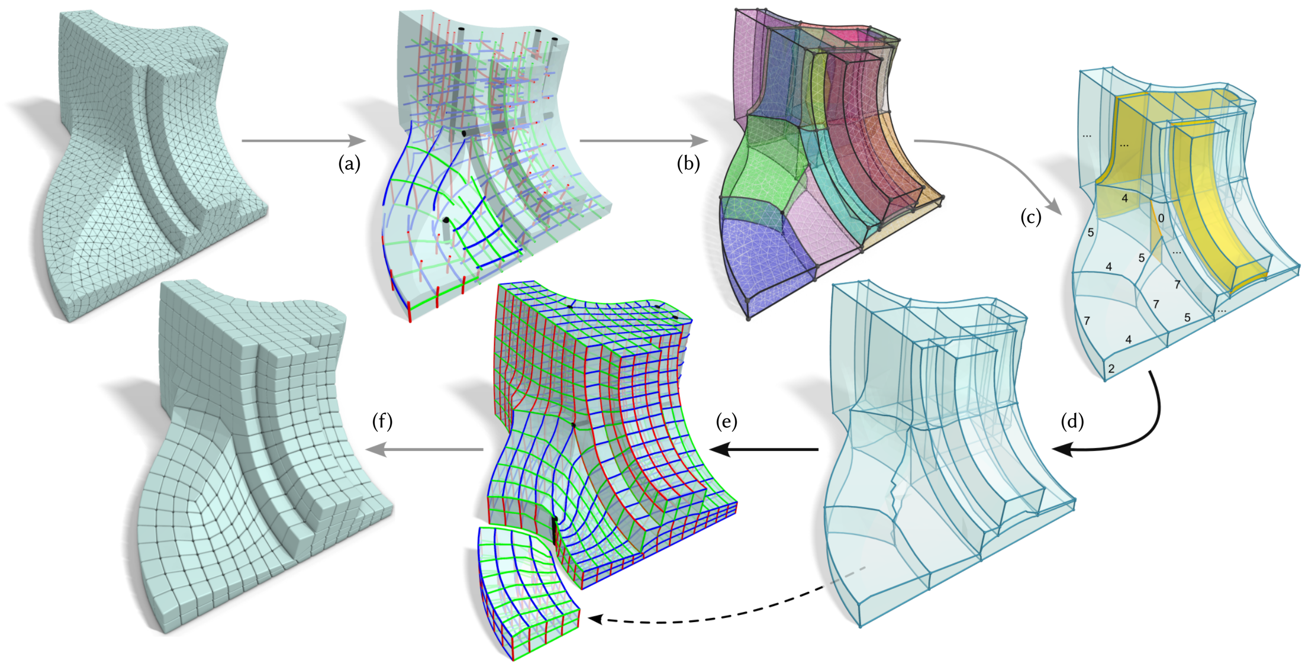


Fig. 2. Overview of a parametrization-based hex meshing pipeline: Given a tetrahedral mesh, **(a)** a global seamless parametrization together with a singularity graph (black) is computed, serving as the guiding structure for **(b)** a parametrization-aligned block decomposition of the input. In a process called quantization **(c)**, elements of this partition are assigned integer target dimensions, including zeros (orange). Our method allows eliminating all zero-elements from the partition, yielding **(d)** a simplified block layout with strictly positive integer dimensions. This facilitates solving the hard problem of computing **(e)** a global integer-grid map respecting these integers by dividing it into simpler per-block cube-map problems. Finally, **(f)** a hexahedral mesh can be extracted as the preimage of the parametric integer-grid, and optionally be further optimized. With our proposed method available (addressing step (d) together with (e)), all steps (b) to (f) can now be considered robustly solved. On the remaining step (a), progress in terms of robustness is ongoing [Liu and Bommes 2023].

decompose this problem—generating an IGM out of a given seamless parametrization—into simpler convex cube-mapping problems, with a reliable solution available. For orientation, Fig. 2 provides an end-to-end overview over an instance of the IGM-based hexahedral meshing pipeline, and shows where our method comes into play.

1.1 Hexahedral Meshing

A detailed recap of IGM-based hexahedral meshing can be found in a recent survey covering this topic [Pietroni et al. 2022]. In a nutshell, a given tetrahedral mesh representation of a 3D object is mapped (using cuts and possibly overlapping charts) into $\mathbb{R}^3$ in such a way that the inverse image of the Cartesian integer-grid forms a hexahedral mesh structure in the object.

A key challenge in this approach is dealing with the discrete degrees of freedom of the problem, ensuring that not partial but only entire grid cells map into the object. Fig. 2a shows a so-called seamless parametrization, in which this is not yet the case; the integer isocurves do not match up across cuts and are not properly aligned with the object's boundary. A first reliable solution to this problem has recently been described [Brückler et al. 2022a], effectively generalizing earlier results for the analogous 2D *quantization* problem of IGM based quadrilateral mesh generation [Campen et al. 2015]. However, this method only reliably *decides* about the discrete degrees of freedom and guarantees feasibility of the chosen integers. Then actually (re)computing the IGM that matches this decision (as shown in Fig. 2e) and therefore properly implies a hexahedral mesh is another problem—for which, so far, only best-effort approaches, based on non-convex numerical optimization, are available. Especially for the interesting use case of generating rather coarse hexahedral meshes—which for instance can serve as a block layout for the generation of finer *block-structured* meshes—failures can be observed for this step of the process.

In the analogous 2D setting, Lyon et al. [2019] have proposed a clever reduction, achieving strong guarantees: Instead of attempting to compute a global IGM subject to challenging constraints, the problem is decomposed into local disc-topology mapping problems to simple square domains—which are easily solved [Tutte 1963].

1.2 Embedded Cell Complexes

Methods building a map from a union of maps, however, require suitable and compatible partitions of both, the map's source and target domain. Such partitions are not easy to construct, already in 2D this requires attention to numerous details [Kraevoy and Sheffer 2004; Schreiner et al. 2004; Lyon et al. 2019]. We exploit the so-called motorcycle complex [Brückler et al. 2022b], cf. Fig. 2b, a structure that proved useful for the above mentioned problem of quantization already [Brückler et al. 2022a], cf. Fig. 2c, and reuse it for our purpose. To that end we interpret it as an *embedded cell complex*, a coarse structure of connected cells (essentially a polyhedral mesh) embedded into the to-be-meshed object (represented by a dense tetrahedral mesh), defining a partition of it into blocks. It is not directly suitable for our purpose, though, but we will show that it can be made suitable, by topologically and geometrically modifying it, as illustrated on a simplistic example in Fig. 3. To this end we introduce a set of collapse operators (together with auxiliary split operators) for volumetric embedded cell complexes, which maintain a consistent embedding. Their strategic application allows us to modify the complex into a state that provides exactly the partition needed for our purpose, cf. Fig. 2d. Namely, by then mapping each block of the partition to a simple axis-aligned cuboid (of an extent derived from the quantization), cf. Fig. 2e, the union of these maps forms an IGM that implies exactly the desired hexahedral mesh connectivity and resolution, as determined by the quantization.

Embedded cell complexes, coarse meshes that are embedded into fine meshes, are actually made use of in various computer graphics contexts (cf. Sec. 2). We therefore formulate our operators rather generically to begin with, abstracting from our specific use case and thereby opening the door for their potential future use in other scenarios as well, and then extend and specialize them to our purpose.

1.3 Contribution

To summarize, we propose to reduce the hard global problem of recomputing a 3D IGM to match integer constraints (in addition to boundary alignment constraints and possibly feature alignment constraints) to simpler local ball-topology mapping problems onto cuboid domains—for which reliable solutions are available [Hinderink and Campen 2023]. To this end we conveniently re-use the embedded cuboidal cell complex (called motorcycle complex or T-mesh) that is made use of in the pipeline (to safely make the integer decisions) anyway. This complex provides a partition and we exploit it to express the mapping problem locally, per block. A key requirement to enable this reduction is a toolbox capable of collapsing elements of this cell complex, while maintaining a valid embedding in the tetrahedral background mesh.

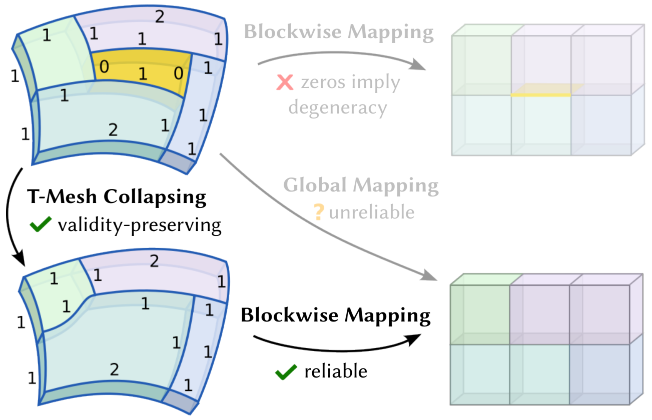


Fig. 3. Illustration of mapping the blocks of an embedded volumetric *T-mesh* (left) onto cuboids (right), of integer extent specified by the quantization. Top: The complex contains a *zero-block*; mapping this block (which has non-zero volume in the object) onto an accordingly sized cuboid (of zero 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: The current state of the art, computing the map not blockwise but *globally* (as in e.g. [Brückler et al. 2022a; Liu and Bommes 2023]), is unreliable, cf. Sec. 8.3.

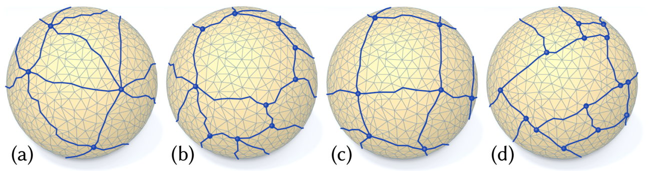


Fig. 4. Illustration of 2D cell complexes embedded in a surface. a) simplicial, b) polygonal, c) quadrilateral, conforming, d) quadrilateral, nonconforming.

After providing a formal description of embedded cell complexes in the volumetric setting (Sec. 3), we therefore introduce a collection of embedding-aware collapse operators for the entities (arcs, patches, blocks) of such complexes (Sec. 4). These preserve overall topology, maintain a valid connectivity, and suitably adjust the geometric embedding in a consistent manner (Sec. 5). We furthermore describe an extension to non-conforming complexes with T-junctions (see Fig. 4 (d) for an illustration in 2D), like the motorcycle complex; this requires the introduction of embedding-aware split operators in addition to the different types of collapses (Sec. 6). These are designed to additionally be capable of maintaining a quantization associated with the cell complex.

We demonstrate the systematic use of this collection of operators in the context of IGM based hexahedral mesh generation, enabling reliable remapping according to any chosen valid quantization (Sec. 7). We furthermore extend this versatile mesh generation pipeline with support for geometric features such as creases on the boundary or material interfaces in the interior. This in particular requires adjustments to our embedding-aware operators, so as to make these feature-aware (App. B).

Method Overview. The input of the hexahedral meshing pipeline from Fig. 2 is a tetrahedral mesh, possibly with attributes indicating desired sizing, orientation, and features; its output is a hexahedral mesh. The steps (d)+(e) we focus on herein take as input a boundary aligned, possibly non-conforming, cuboidal block decomposition together with quantization and feature information. Their output is an IGM, a locally injective feature-aligned integer grid map. Step (d) consists of algorithmically transforming the input block decomposition into a state such that the mapping problem can be expressed blockwise in step (e). Concretely, we interpret the decomposition as a cell complex embedded in a tetrahedral background mesh, and incrementally apply collapse operators to get rid of all cells of zero length, area, or volume, as specified by the input quantization information, cf. Fig. 3. The operators are designed to always maintain a consistent bijective embedding of the cell complex in the background mesh, while respecting boundaries, singularities, and features.

2 RELATED WORK

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]. So-called *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 [Kraevoy et al. 2003; Praun et al. 2001]. Also polygonal [Tong et al. 2006; Born et al. 2021] and non-conforming meta meshes (with T-joints) [Nuvoli et al. 2019; Lyon et al. 2019, 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. 4 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] and non-conforming [Brückler et al. 2022b,a] varieties. So-called base complexes of hexahedral meshes [Gao et al. 2015, 2017; Gunpinar et al. 2023] can be viewed as meta meshes embedded in a hexahedral background mesh.

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 [Botsch and Kobbelt 2004; Freitag and Ollivier-Gooch 1997; Trotts et al. 1998; Daniels et al. 2009; Peng et al. 2011; Shen et al. 2021; Gao et al. 2015], embedded meta meshes require an accompanying update of the discrete embedding, so as to keep it consistent with the connectivity. 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.

Integer-Grid Maps. Semi-structured quadrilateral mesh generation based on global surface parametrization goes back to works such as [Bommes et al. 2009; Kälberer et al. 2007; Tong et al. 2006]. The class of parametrizations that induce boundary-conforming pure quad meshes was termed integer-grid maps [Bommes et al. 2013]. A relaxation of this class are so-called seamless maps [Myles and Zorin 2012; Campen et al. 2019; Levi 2021; Shen et al. 2022]. They essentially ignore the integer constraints that need to be met by integer-grid maps, treating discrete degrees of freedom as continuous. Such maps are often generated as a precursor to guide the choice of integers for IGMs in a process called quantization [Campen et al. 2015; Bommes et al. 2013; Kälberer et al. 2007; Lyon et al. 2019].

All these concepts have been extended from the 2D surface setting to the 3D volume setting, so as to facilitate hexahedral mesh generation [Nieser et al. 2011; Li et al. 2012; Jiang et al. 2014; Liu et al. 2018; Brückler et al. 2022b,a; Liu and Bommes 2023; Cherchi et al. 2016; Lyon et al. 2016].

Embedded cell complexes are employed in this context in particular when it comes to the sub-problem of quantization [Campen et al. 2015; Lyon et al. 2019; Pietroni et al. 2021; Brückler et al. 2022a]. In one case [Lyon et al. 2019] embedding-maintaining meta mesh modification operators are used, albeit only in a 2D surface setting.

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], a recent progressive approach [Nigolian et al. 2023], 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) the above simpler mapping problem.

3 BACKGROUND

We are going to work with 3D objects represented by a tetrahedral mesh. A tetrahedral mesh $\mathcal{M} = C \cup F \cup E \cup V$ is a finite three-dimensional pure simplicial complex (cf. [Klette 2000]) consisting of tetrahedra $C = \{c_1, c_2, \dots\}$, facets $F = \{f_1, f_2, \dots\}$, edges $E = \{e_1, e_2, \dots\}$, and vertices $V = \{v_1, v_2, \dots\}$. Its boundary $\partial\mathcal{M}$ is a triangle mesh, consisting of entities from F, E, and V. The mesh's geometric realization is assumed to be piecewise linear, thus fully defined by vertex coordinates $V \to \mathbb{R}^3$. For notational precision, we distinguish between an abstract element of $\mathcal{M}$ and its concrete geometric realization: For a vertex $x \in V$, we let $[x]$ denote the point in $\mathbb{R}^3$ it occupies. Similarly, for an edge $x \in E$, $[x]$ is an open line segment, for a facet $x \in F$ an open triangle, and for a tetrahedron $x \in C$ an open tetrahedron. 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 will serve as basis to define the embedding of a cell complex $\mathcal{S}$ (the *meta mesh*).

3.1 Cell Complex

For our purposes, $\mathcal{S} = B \cup P \cup A \cup N$ is a finite pure polyhedral cell complex, consisting of blocks (3-cells) $B = \{b_1, b_2, \dots\}$, patches (2-cells) $P = \{p_1, p_2, \dots\}$, arcs (1-cells) $A = \{a_1, a_2, \dots\}$, and nodes (0-cells) $N = \{n_1, n_2, \dots\}$. The blocks are not simplices, but may be arbitrary topological polyhedra.

We define $\dim(s) = k$ if s is a k-cell. Two cells s_i, s_j with $|\dim(s_i) - \dim(s_j)| = 1$ can be either incident, denoted as $s_i \leftrightarrow s_j$, or non-incident. We also define a directed incidence relation: $s_i \rightarrow s_j$ iff $s_i \leftrightarrow s_j \wedge \dim(s_i) < \dim(s_j)$. This gives rise to a graph based cell connectivity visualization as used in Fig. 6. A *link* between any two elements $s_0, s_n \in \mathcal{S}$ is a sequence of elements $s_0, s_1, \ldots, s_n$ such that $s_i \leftrightarrow s_{i+1} \forall i$. We call two elements *connected* if there is a link between them and a subset $\mathcal{S}' \subseteq \mathcal{S}$ is called connected, if all elements in $\mathcal{S}'$ are pairwise connected within $\mathcal{S}'$. A link is a *downlink* (*uplink*) if $\dim(s_i)$ is monotonically decreasing (increasing). For any cell s its *closure* $\langle s \rangle$ is the union of s with all cells for which a downlink from s exists. The closure of a set of cells is defined as the union of the closures of each cell. The *boundary* of a cell is defined as $\partial s = \langle s \rangle \backslash s$, i.e. the set of all cells reachable from s via a downlink.

Note that we make no restrictive assumptions regarding *self-adjacency*; a cell may be linked to itself via one of the cells from its boundary. For instance, a block may be self-adjacent via a patch, an arc, or a node.

3.2 Embedded Cell Complex

A discrete cell complex embedding of $\mathcal{S}$ into tetrahedral mesh $\mathcal{M}$ is a map $\mathcal{I} : \mathcal{S} \hookrightarrow 2^{\mathcal{M}}$, mapping each cell $s \in \mathcal{S}$ onto a *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:

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

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 $\mathcal{I}^*$ that additionally includes all interior lower-dimensional elements in a cell's image: $\mathcal{I}^*(s) := \langle \mathcal{I}(s) \rangle \setminus \langle \mathcal{I}(\partial s) \rangle$. This also directly defines a cell's geometry, via $[s] := [\mathcal{I}^*(s)]$, i.e. the geometric realization of cell s is defined by that of the elements it is embedded in. This is illustrated in Fig. 5.

The following conditions need to be met by all cells to make the map $\mathcal{I}$ an embedding:

- *Injectivity:* $\mathcal{I}^*(s_i) \cap \mathcal{I}^*(s_j) = \varnothing$ for all $s_i \neq s_j \in \mathcal{S}$.
- *Structure Preservation:* $[\partial s] = \partial[s]$.
- *Manifoldness:* $\dim(s) = k > 0 \Rightarrow [s]$ is an open k-ball.

Intuitively, the embedding is structure preserving if for each cell the 'image of its boundary' equals the 'boundary of its image'. For many use cases, including ours, the embedding is furthermore assumed to be *surjective*, i.e. all of $\mathcal{M}$ is covered by $\mathcal{S}$: $\bigcup_{s \in \mathcal{S}} \mathcal{I}^*(s) = \mathcal{M}$.

The embedded cell complexes employed in the applications discussed in Sec. 2 typically meet these conditions by construction. The modification operators we introduce are designed to maintain a valid embedding throughout. Whenever the complex is modified on the abstract incidence level, the embedding is updated consistently, preserving the satisfaction of all of the above conditions.

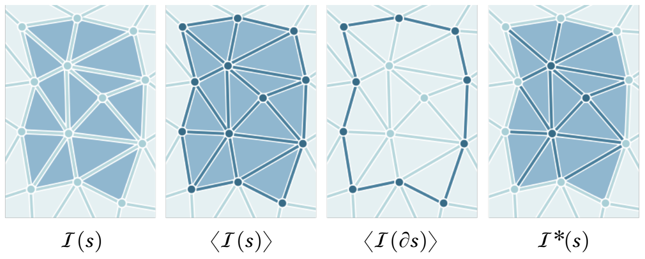

$\mathcal{I}(s)$
$\langle \mathcal{I}(s) \rangle$
$\langle \mathcal{I}(\partial s) \rangle$
$\mathcal{I}^*(s)$

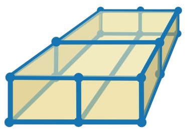
Fig. 5. Illustration of a discrete embedding $\mathcal{I}(s)$ of a quadrilateral patch s (left) and its implied geometric realization $[s] = [\mathcal{I}^*(s)]$ (right), topologically an open disk. In the center two intermediate definitions are illustrated.

Self-Adjacency. It is important to note that while $[s]$ is an (open) k-ball, its closure is not necessarily a (closed) k-ball, namely whenever cells are self-adjacent in some way. 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). We therefore keep definitions this general.

Inverse Embedding. Note that disjointness allows us to define a kind of inverse: we write $\overleftarrow{\mathcal{I}}(x) = s$ if $x \in \mathcal{I}(s)$, and $\overleftarrow{\mathcal{I}}(x) = 0$ if x is not part of any cell's embedding. This is also convenient for implementation purposes (and is therefore made use of in Sec. 5) as it, in contrast to $\mathcal{I}$, does not require a set-valued representation per element. For brevity, we will use the notation $\mathcal{I}_s := \mathcal{I}(s)$ and $\overleftarrow{\mathcal{I}}_x := \overleftarrow{\mathcal{I}}(x)$ in the following.

3.3 Cubical Complex

Of particular interest in the context of hexahedral mesh generation are embedded cell complexes with blocks that are structurally cubes, consisting of six sides and eight corners. We can further distinguish conforming and non-conforming cubical complexes. In the former case, each cube side is formed by one patch, each cube edge by one arc. In the latter case, rather, each cube side may be formed by a connected set of multiple patches, each cube edge by a sequence of arcs, as illustrated here (and in 2D in Fig. 4d). Arcs between patches of the same cube side are referred to as T-junctions in this context, the complex as a whole as T-mesh.



Motorcycle Complex. A particular method to generate an embedded cubical complex is by means of tracing iso-surfaces in volumetric seamless parametrizations [Nieser et al. 2011]. Brückler et al. [2022b] describe such a construction, the so-called motorcycle complex, a generalization of the motorcycle graph [Eppstein et al. 2008] from surfaces [Campen et al. 2015; Lyon et al. 2021a] to the volume. The result is a non-conforming complex, i.e. a T-mesh. In this case the blocks of the complex are not just structurally cubical, but they are geometrically cuboids with respect to the underlying parametrization, in particular axis-aligned cuboids in parameter space.

T-Mesh Quantization. In the context of the hexahedral mesh generation method of Brückler et al. [2022a], a quantization is associated with such a T-mesh. This is the assignment of an integer value to each arc, subject to the condition that each of the six sides of each

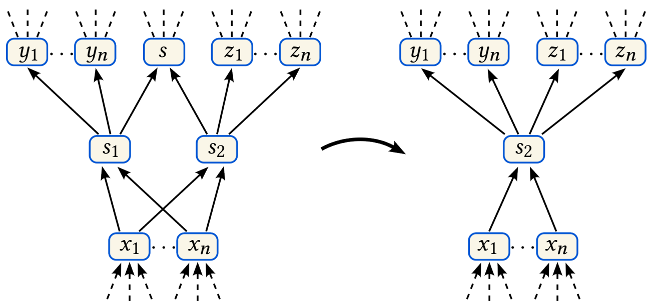


Fig. 6. Illustration of a collapse operation (of cell s) in terms of the incidence relation graph, showing the state before (left) and after (right).

block are rectangular (and thus the block cuboidal) when interpreting these integer values as the arcs' lengths. Such a quantization directly implies a regular $m \times n \times o$ grid of hexahedra per block, in such a way that they stitch together globally to a conforming hexahedral mesh—at least structurally; determining a suitable or just valid geometric embedding of this mesh is another question.

Importantly, a quantization's integer values are not necessarily positive. Zero values may be assigned to some arcs. This is important to achieve high structural and geometric mesh quality, especially when aiming for rather coarse or block-structured hexahedral meshes [Brückler et al. 2022a]. Therefore implied arc lengths, patch areas, and block volumes can be zero, i.e. they imply and correspond to zero hexahedral mesh edges, faces, or hexahedra, respectively. This precludes the approach of simply, per block, mapping the corresponding regular grid of hexahedra into the block to obtain a valid geometric embedding, cf. Fig. 3. However, by first collapsing all elements with assigned zero extent, we can arrive at an embedded T-mesh with a quantization free of zeroes. This then enables the per block approach of embedding the hexahedral mesh, or analogously remapping an IGM.

4 COLLAPSE OPERATORS

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. We first consider the abstract connectivity aspects of such collapses. Afterwards, in Sec. 5, the accompanying treatment of the complex's embedding in a background mesh is addressed.

Concretely, here we describe the following atomic operators:

- collapsing a single arc, incident to one or two nodes;
- collapsing a single patch, incident to one or two arcs;
- collapsing a single block, incident to one or 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. In Sec. 6 we describe collapse operators (as an extension of the base operators introduced here) also for more general patches and blocks, in a setting where additional information (an associated quantization) disambiguates the situation.

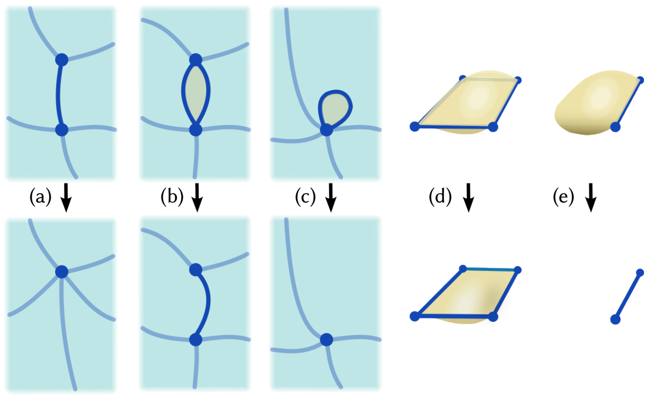


Fig. 7. Illustration of several types of collapses. a) collapse of arc with two nodes. b) collapse of patch with two arcs. c) collapse of arc with one node; note that the yellow patch becomes isolated and is deleted. d) collapse of block with two patches. e) collapse of patch with one arc; note that the contained block becomes isolated and is deleted.

4.1 Connectivity Updates

We exploit a symmetry among the above three operators to simplify exposition: In each case, an i-dimensional cell s incident to one or two $(i-1)$-dimensional cells is to be collapsed. Fig. 6 illustrates this situation (for the case of *two* incident cells, s_1 and s_2) using the relevant excerpt from the incidence graph defined by the cell complex's incidence relation $s_i \to s_j$ (cf. Sec. 3.1).

The collapse of cell $s \in \mathcal{S}$ with $\dim(s) > 0$ is performed as follows: If there are two cells, s_1 and s_2, with $s_i \to s$, then s can be collapsed if $x \to s_1 \Leftrightarrow x \to s_2$, i.e. s_1 and s_2 have the same boundary. A collapse is a directed operation, i.e. we can choose whether s_1 or s_2 vanishes. We assume s_1 is supposed to vanish in the following. If there is only one cell $s' \to s$, we denote $s_1 = s_2 = s'$ to unify exposition.

The following updates of the incidence relation are performed to execute the collapse of s:

(1) For each cell x, if $s_1 \to x$, then set $s_2 \to x$.
(2) Remove s from $\mathcal{S}$.
(3) If $s_1 \neq s_2$, remove s_1 from $\mathcal{S}$.

When removing a cell, all incidence information associated with it is implicitly deleted. Finally, all cells x that became isolated (i.e. there no longer is any cell y such that $x \to y$) are deleted. Fig. 6 depicts this update of the incidence relation graphically. Fig. 7 shows a geometric illustration of the different cases of collapses.

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 cause that boundary to become non-manifold. Also non-pure complexes can 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; the complex remains in the realm of manifoldness and pureness.

5 EMBEDDING UPDATES

When collapsing cells of complexes that are embedded, the embedding needs to be updated in addition to the connectivity updates addressed in Sec. 4. Otherwise, when just restricting the map $\mathcal{I}$ to the remaining cells, it would no longer satisfy the embedding conditions stated in Sec. 3.2. In particular, structure-preservation and surjectivity are at stake.

In this section we describe, for each collapse operator, the discrete embedding updates to be performed in conjunction with the connectivity changes, so as to maintain a valid embedding. Conceptually, when a cell is removed due to a collapse, the embedding update's main goal is to shrink the cell's image under the embedding map to zero extent, while adjusting the surrounding cells' images to take up this space and reestablish a structure-preserving state. In this way consistency between connectivity and geometry is recovered.

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 transforming the former embedding using a set of suitable operators.

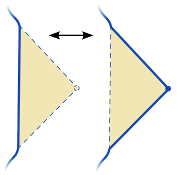
Note that the set of possible structural configurations around a to-be-collapsed cell is very diverse. The number of patches incident on an arc is varying, the number and adjacency pattern of arcs incident on a node is varying, subsets of incident cells can lie in the boundary, cells can be self-adjacent 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*, along a cell's uplinks. 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.

For simplicity, we will not explicitly discern between a cell and its embedding image in this section, 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.

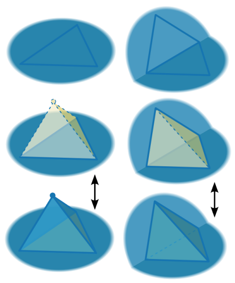
5.1 Base Operators

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. It will rather be used in Sec. 5.2 to compile 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. Care needs to be taken to avoid interference with other arcs or patches possibly touching the edges to be inserted into a; this is done by local refinement of the background mesh, as detailed in App. A.2.



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. Aditionally, 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. In this case as well, background mesh refinement (App. A.2) may be needed to avoid overlaps with arcs or other patches possibly touching the facets to be inserted into p.



5.2 Main Operators

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 uplinks 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).

5.2.1 *ArcPatchShift.*
Let us first consider only a single patch p incident to the arc a to be shifted, as displayed in Fig. 8. Initially a is

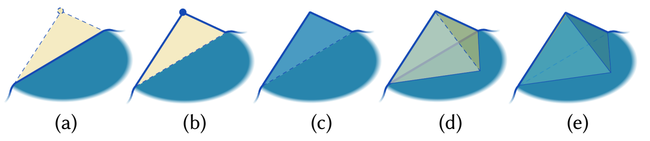


Fig. 8. Illustration of sub-steps of (part of) a ArcPatchShift. After an arc (dark blue) is shifted across a single facet (beige) via an ArcShift, one patch (blue) incident to the arc is first reconnected to the shifted arc by appending the traversed triangle into the patch. Then it is lifted off by (one or more, here one) PatchShifts to resolve overlaps with other patches incident to the arc. Fig. 9 illustrates how multiple incident patches are handled sequentially.

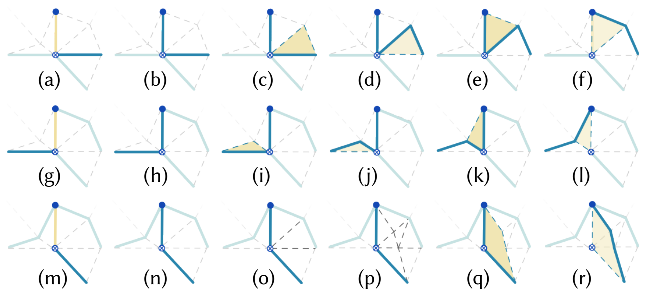


Fig. 9. Cross-section illustration of a ArcPatchShift. The first row corresponds to Fig. 8, except that here two PatchShifts are required to lift off the first incident patch. The further rows show the handling of the second and third incident patch. Note that in the end the surrounding wheel of patches of the shifted arc is properly connected to it again.

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. 8, 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. 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 pointing down-right in Fig. 9 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. 9o, with Fig. 9p indicating the local background mesh refinement necessary to allow the last patch to be shifted. This is discussed in detail in App. A.2. Note that the last patch shift in Fig. 9p-r may be skipped altogether, as Fig. 9o is already a valid ending configuration; this is discussed further in App. A.1.

Alg. 1 in App. A describes the procedure in a formal way.

5.2.2 *NodeArcPatchShift.*

Consider the situation in Fig. 10a: A node, with three incident arcs and three incident patches 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. 10, one such arc is chosen and arc-node connectivity recovered by appending the traversed edge into this arc (Fig. 10c-d). Using a sequence of ArcPatchShifts along a triangle fan (Fig. 10e) 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 repeated for any arcs remaining in the current layer (Fig. 10g-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. 10k-n).

While Fig. 10 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.

Alg. 2 in App. A describes the behaviour in a formalized way.

5.3 Embedded Collapse Operators

Using merely the above two operators, NodeArcPatchShift and ArcPatchShift, we can now define the embedding updates to accompany the different types of collapses.

Embedded Arc 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. Fig. 11 shows such a collapse fo an arc composed of three edges.

Embedded Patch 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_1 of the patch is chosen, its embedding erased, and then the remaining arc a_2 is incrementally shifted across p by a sequence of ArcPatchShifts: any facet of $\mathcal{I}_p$ incident to at least one edge $\mathcal{I}_{a_2}$ is chosen, a is shifted across that facet and the process is repeated until $\mathcal{I}_p = \emptyset$ and $\mathcal{I}_{a_2}$ coincides

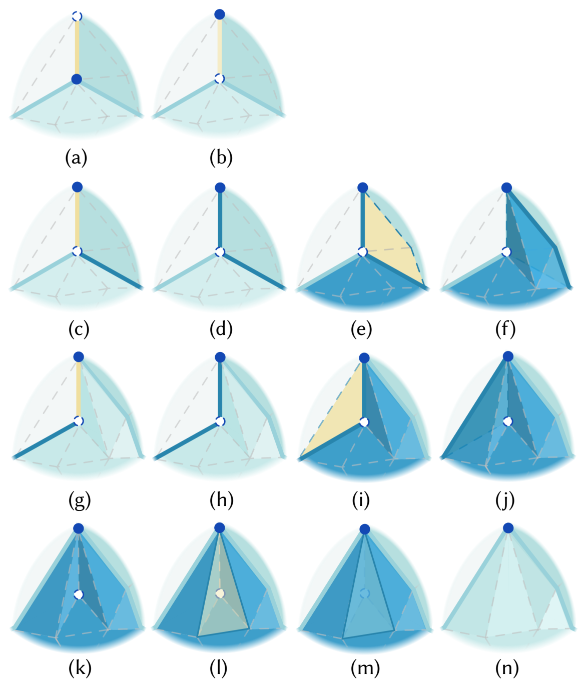


Fig. 10. Illustration of sub-steps of (part of) a NodeArcPatchShift. Arc by arc, ArcPatchShift operators are applied to reconnect these arcs with a shifted incident node. In the end (last row) a PatchShift is applied to lift the blue patch off of the former node vertex (white).

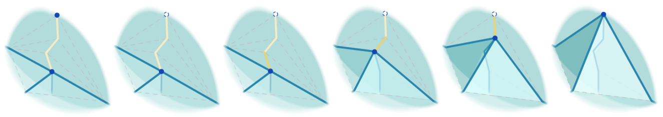


Fig. 11. When topologically collapsing an arc (yellow) the complex's embedding can be updated through (after deleting one incident node from the embedding map) a sequence of NodeArcPatchShifts, edge by edge.

with the former $\mathcal{I}_{a_1}$. This process is demonstrated in Fig. 12 for a case where there is only one patch dragged with the shifting arc and with shifts applied batch-wise for brevity. It is sensible in this 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 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].

Embedded Block Collapse. While the collapse of a block b with one or two incident patches with common boundary 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.

Embedded collapse operators for loop arcs with a single node and patches with a single arc can be defined using our fundamental operators as well (e.g. contracting a loop arc onto its node using ArcPatchShifts across a contained patch or surface). Such special cases, however, are not relevant for our intended application.

Boundaries. Minor special care that needs to be taken for cells embedded in the boundary is detailed in App. A.1.

Geometric Degrees of Freedom. While the effect of a collapse on the complex's connectivity is uniquely defined (cf. Sec. 4.1), geometrically the embedding update comes with degrees of freedom. The above described embedded collapse operators 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

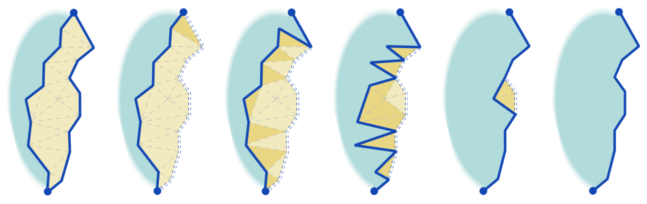


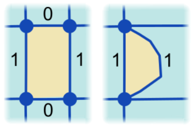
Fig. 12. The embedding update for a pillow patch collapse is performed (after deleting one incident arc from the embedding map) by a sequence of ArcPatchShifts, across all triangles of the patch (yellow).

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, cf. Sec. 6.2.2.

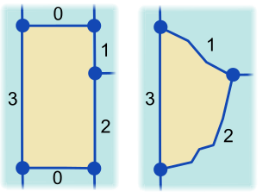
6 T-MESH COLLAPSES

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 patches with more than two incident arcs and for 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 Sec. 3.3. Concretely, we define a collapse operator for patches that have two sides, two *logical arcs*, that may each consist of multiple arcs, and a collapse operator for blocks that have two sides, two *logical patches*, that may each consists of multiple patches. The division into multiple arcs and patches is due to T-junctions, induced by other arcs or patches that are incident to the interior of a logical arc (patch) from outside the patch (block).

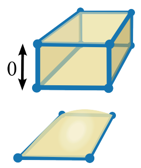
Challenges of Non-Conformity. These operators are particularly relevant for our use case (Sec. 7). In that context, arcs have assigned virtual lengths (from a quantization), which also induce areas of patches and volumes of blocks. We will see that the main goal is to collapse all cells that have extent zero in this sense. In a conforming cubical complex, this is easily possible with the operators from the previous section, as illustrated in the inset. First, zero-arcs can be collapsed. This will turn zero-patches into pillow-patches, which can then be easily collapsed.



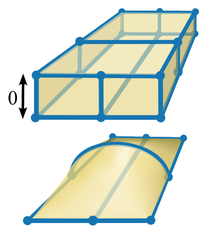
In a T-mesh however, collapsing zero-arcs can yield a *quasi*-pillow patch—a patch with two sides (note that the right node is a T-junction w.r.t. the patch), but more than two arcs. The fact that this patch, however, should be collapsed (it is a zero-patch) illustrates the need for the above mentioned generalized collapse operators.



Analogously, a block in a conforming cubical complex is incident to exactly six patches. For a block with zero-arcs along one dimension, collapsing the four zero-arcs and the resulting four pillow-patches always yields a pillow-block, allowing blocks as well to be fully collapsed using the operators from the previous section.

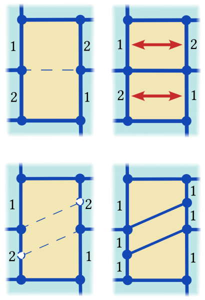


In a T-mesh, by contrast, the six sides of a block may in total contain more than six patches, and collapsing all zero-arcs of a block does not yield a pillow-block. Even if all patches of the four collapsed sides were conforming initially, i.e. composed of exactly four arcs each, and as such could be handled by standard pillow-patch collapses after collapsing the arcs, the remaining *quasi*-pillow block could not be collapsed using known operators.

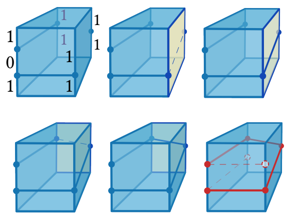


Reduction to Conformity? It is tempting to try to reduce the non-conforming situation to a conforming one. Then the known operators could be applied. This, however, is not easily possible in general, especially when the complex needs to remain consistent with the assigned quantization, as the following 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.



6.1 Bisection Operators

For these reasons, we introduce additional operators to split, or more precisely bisect, 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.

6.1.1 Patch Bisection. In the following we will refer to a patch that only has two opposite sides but is incident to more than two arcs as a *quasi-pillow* patch. We handle such 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. Fig. 13 illustrates this.

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. 13) 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. 13c,f). As discussed above, this arc or node may not be unique if there are

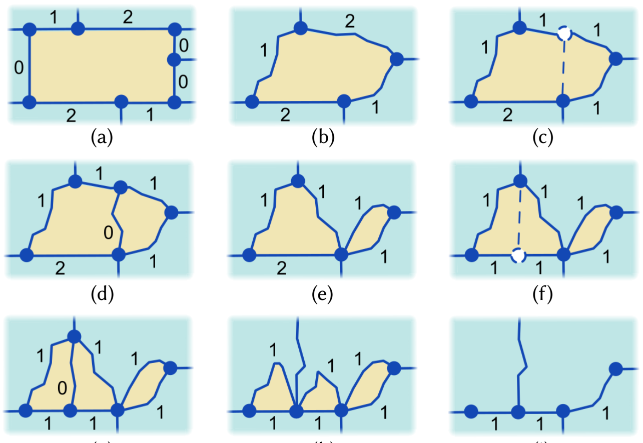


Fig. 13. From an initial configuration (a), a quasi-pillow patch results after performing zero-arc collapses (b). This allows it to be collapsed by means of simple patch collapses (h-i), after recursively bisecting it (d,g) by extending a T-junction (c,f).

zero-arcs on the opposite side. We say a patch bisection operation is *executable* if no such zero-arcs cause an ambiguity.

Alg. 3 in App. A formalizes this patch bisection operator.

6.1.2 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. 14a) 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 (Fig. 14a). We extend this T-junction across the subsequent patch on the block boundary (analogous to the above patch bisection case, respecting the quantization). This is repeated until the arc cycle is closed (Fig. 14b-c).

Once this cycle of arcs is determined, we can insert a new patch, bisecting the block, incident to this boundary cycle (Fig. 14d). 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.

Alg. 4 in App. A details this process. Again, we call a block bisection *executable* if it involves no ambiguities due to zero-elements.

6.2 Overall Collapsing 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

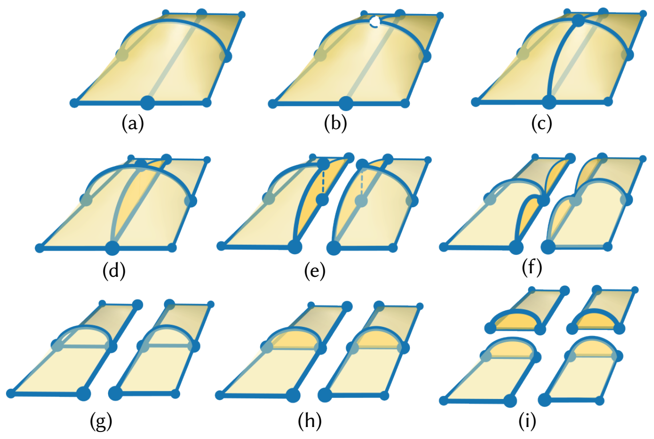


Fig. 14. A pillow patch (a) is bisected by extending a T-arc (here on the bottom side) around the block, patch by patch (b) until it forms a cycle (c). The a patch is inserted, bounded by this cycle, splitting the block into two (e). Here quasi-pillow-patch collapses (e-f) and then pillow-patch collapses (f-g) become possible. Recursive application of block bisection (g-i) in the end leaves simple pillow-blocks.

cells can lead to special configurations (such as zero-arcs forming a loop, zero-patches whose boundary is a single arc or node); this would require the definition and implementation of additional embedding-maintaining collapse operators for these special 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. Then, e.g., a cycle of two zero-arcs will be treated by means of a pillow-patch collapse (if it bounds a patch) or a block bisection (otherwise), before one of these zero-arcs would be collapsed leaving a zero-loop-arc.

This is achieved by applying the basic embedding-maintaining operators introduced in Sec. 5.3 and Sec. 6.1 to all zero-elements in a specific order:

- Repeat until no zero-elements 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 were determined by a *valid* quantization, all zero-elements will eventually be collapsed. This is due to the absence of paths of zero-elements between critical elements such as boundaries [Brückler et al. 2022a].

6.2.1 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 blockwise 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.

6.2.2 Geometry. 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.

7 HEXAHEDRAL MESHING

We now turn to the application scenario of parametrization based hexahedral mesh generation—our original motivation for considering collapses on embedded cell complexes. As indicated in Sec. 1, we will make use of the results of the previous sections to reduce a very hard subproblem of the meshing process—with no reliable solution in sight—to multiple smaller and simpler problems.

7.1 IGM Based Meshing Pipeline

The generic integer-grid map based pipeline for the generation of semi-structured quadrilateral or hexahedral meshes, already hinted at in Sec. 1.1, goes back to [Kälberer et al. 2007; Bommes et al. 2009] and was later refined, improved, and extended in a variety of works [Pietroni et al. 2022]. Its main steps (cf. Fig. 2), given an input object to be meshed, are:

(1) Compute a global seamless map, typically topologically and geometrically guided by a boundary-aligned frame field.
(2) Quantize the seamless map, i.e. settle the integer degrees of freedom.
(3) Recompute a global seamless map, subject to constraints reflecting the integer choice, making it an IGM.
(4) Extract the quad/hex mesh implied by the IGM.

For the 2D case (quad meshes in the plane or on surfaces), reliable solutions for every step are available by now. For the 3D case (hex meshes in the volume), reliable solutions are available for step (2)

[Brückler et al. 2022a] and step (4) [Lyon et al. 2016]. Step (1) is a topic of active research, where continuing progress can be witnessed [Viertel et al. 2016; Liu et al. 2018; Reberol et al. 2019; Corman and Crane 2019; Palmer et al. 2020; Pietroni et al. 2022]; main challenges yet lie in the larger topological gap between frame fields and seamless maps in 3D (*meshability* issue). Progress on this front has very recently been reported [Liu and Bommes 2023].

Step (3) lacks a reliable solution so far. Therefore, even in very recent work [Brückler et al. 2022a; Liu and Bommes 2023] a best-effort technique is employed for step (3)—essentially using a non-convex constrained numerical optimization formulation. Its success rate depends on the targeted mesh resolution, where a coarser target resolution implies a harder problem, cf. Sec. 8.3.

Using the collection of operators for embedded volumetric complexes introduced in the previous chapters, we are able to reduce step (3) to a problem that is actually simpler (rather than harder) than step (1). In fact, in step (1) a volumetric chart-based, self-overlapping but locally injective map, with restricted transition functions across cuts and a (frame field implied) network of singularity points and curves, needs to be computed. We, by contrast, reduce step (3) to standard mapping problems of ball-topology regions onto simple convex domains (cuboids), without any cuts, transitions, or singularities. This is made possible by exploiting the fact that step (3) is a *re*-computation problem. In brief, we exploit the result of step (1) (or a structure derived from that) to enable this reduction.

7.2 Blockwise Remapping

A 3D object equipped with a volumetric seamless map can be partitioned into a (conforming or non-conforming) cubical complex that is parametrically aligned with the map, e.g. by means of the motorcycle complex [Brückler et al. 2022b]. Under the map, each block is an axis-aligned cuboid. In the above mentioned reliable solution for step (2), this complex is generated anyway. It is embedded in a tetrahedral mesh of the given object. On the basis of this complex, the task of recomputing the map, subject to the above mentioned constraints, step (3), can be expressed as a blockwise problem. In essence, each block (of ball-topology) just needs to be remapped onto a cuboid again, albeit of different extent than in the original map. The extent is given by the quantization determined in step (2), as also discussed briefly in [Brückler et al. 2022b, §7.1].

However, this blockwise perspective of the problem is only feasible if no block of the complex is assigned a target extent of zero (in one or more dimensions) by the quantization. Mapping a block onto a cuboid of size zero would necessarily force the map into degeneration, as illustrated in Fig. 3, i.e. the blockwise perspective is unsuitable in this case. Using only strictly positive quantizations restricts the solution space significantly, excludes desirable quantizations, and lowers output mesh quality, as discussed in detail by Brückler et al. [2022a]. Therefore we wish to support zero values. Note that the reparametrization problem is still feasible in such a case, just the blockwise approach is unfit.

Our solution to this issue is to collapse all the zero-elements in the complex—by applying the operators introduced in the previous section. In the analogous 2D setting, relevant for quad mesh generation, such a strategy is followed by Lyon et al. [2019]. We now address the 3D setting using our collection of novel operators. In fact, prepared with these operators, all we need to do is apply them to the embedded cubical complex, in the order described in Sec. 6.2, collapsing zero-elements until none are left. The quantization is carried over to the reduced complex as described in Sec. 6. This yields a new embedded cubical complex, with an associated quantization in which now all extents are strictly positive. Hence, the blockwise remapping approach is feasible, as in Fig. 3 bottom.

7.3 Singularities & Features

A little additional care is necessary when applying the collapses in this specific setting: Some elements of the complex may coincide with singularities by construction. Their embedding is (or can be) crucial and must not be changed (except *within* a singularity itself). The same holds for *feature* curves or surfaces, marked in the input object, that the generated mesh should align to. Such feature constraints are not supported by the cubical complex construction [Brückler et al. 2022b] and the quantization computation [Brückler et al. 2022a]. We describe an extension of these to add support for feature points, curves, and surfaces in App. B.

Recall that in the collapse operators introduced there is a *direction* degree of freedom: Which of the two incident lower dimensional cells vanishes and which one remains (cf. Sec. 4.1). When collapsing a cell where one of the two is marked (as a singularity or a feature), we just always need to choose this one to remain, keeping its embedding unchanged. This is analogous to collapses at the boundary, as detailed in App. A.1. 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 *separation* conditions [Brückler et al. 2022a].

For collapses of arcs or patches that are marked (as a singularity or a feature) themselves, the same rules described for collapses of boundary arcs and patches in App. A.1 need to be applied. This ensures that the embedding of such arcs and patches is not lifted off of the intended singularity or feature.

7.4 Mapping and Meshing

After collapsing all zero-elements of the complex (and optimizing the embedding as discussed in Sec. 6.2.2), we are left with a complex such that for each block $b \in B$ the associated quantization defines a positive extent $m_b \times n_b \times o_b \in \mathbb{Z}^3$.

This leads to a mapping problem per block, computing a map $\phi_b : [\langle \mathcal{I}_b \rangle] \to D_b$, i.e. from the geometric extent of the image of b (the region of $\mathcal{M}$ that b is embedded into) 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 $\partial\phi_b : \partial[\langle \mathcal{I}_b \rangle] \to \partial D_b$ needs to be prescribed to ensure compatibility with adjacent blocks. Concretely, note that the block's boundary is partitioned into patches. Each patch p has a target extent $m_p \times n_p$ assigned by the quantization. The boundary map $\partial\phi_b$ is chosen such that each such patch maps onto an axis-aligned rectangle of size $m_p \times n_p$ in ∂D_b. The interior of a patch can be mapped into this rectangle arbitrarily; this just needs to be done the same way for both blocks that are incident on the patch. The union of these maps ϕ_b defines an IGM for $\mathcal{M}$, namely $\phi = \bigcup_{b \in B} \phi_b$. Note that the patches shared by adjacent blocks, due

Table 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 the number of cells. Models for which the coarsest quantization is strictly non-zero are omitted.

Model	$\lvert B_0\rvert$		$\lvert P_0\rvert$		$\lvert A_0\rvert$		$\frac{\lvert S_0\rvert}{\lvert S\rvert}$	$\updownarrow_B$	$\updownarrow_P$	$\updownarrow_A$	$\vert_B$	$\vert_P$	$\sum$	$\Delta\lvert S\rvert$
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%

to being mapped compatibly by prescribing the boundary map per block, ensure that the transition functions of ϕ between adjacent blocks are integer-grid automorphisms [Bommes et al. 2013], as required for an IGM.

Such a mapping problem, from a ball-topology region to a convex region, without cuts, transitions, singularities, overlaps, and with fixed boundary, is more standard and arguably simpler than a global volumetric IGM mapping problem with free boundary and constrained transitions, interior pin constraints, etc. The recent method of Hinderink and Campen [2023] provides a reliable solution, supporting boundary constraints and guaranteeing bijectivity. 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. 20). Due to this optimization, across block boundaries, the concrete geometric embedding of the collapsed complex (used merely for initialization) is of low relevance.

8 RESULTS

We have implemented the operators, embedding updates, and the collapsing strategy described in Secs. 4 to 6 as well as the blockwise remapping strategy described in Sec. 7 in C++, using the OpenVolumeMesh data structure [Kremer et al. 2013]. Code is available at github.com/HendrikBrueckler/C4HexMeshing.

8.1 Datasets

As input data for the purpose of testing and evaluation we use the following three sets of instances: The two result datasets of [Brückler et al. 2022a], containing motorcycle complexes on models from two different sources. Furthermore, as these do not contain any feature information, we create a third set of instances by computing motorcycle complexes (taking features into account, cf. App. B) on models from the HexMe dataset [Beaufort et al. 2022], which comes with prescribed feature points, curves, and surfaces. These datasets then contain 15, 82, and 110 instances, respectively, in total: 207.

On these, quantizations of varying resolution are computed using the code published by Brückler et al. [2022a]. In particular, to maximally challenge our method, for the stress test in Sec. 8.2 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 Sec. 8.3 we furthermore generate sequences of increasingly fine quantizations. Finally, for the purpose of generating block-structured hexahedral meshes in Sec. 8.5, we generate a moderately coarse quantization to determine a *block layout*.

8.2 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 Sec. 6.2, 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,
- 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 Tables 1, 2 and 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

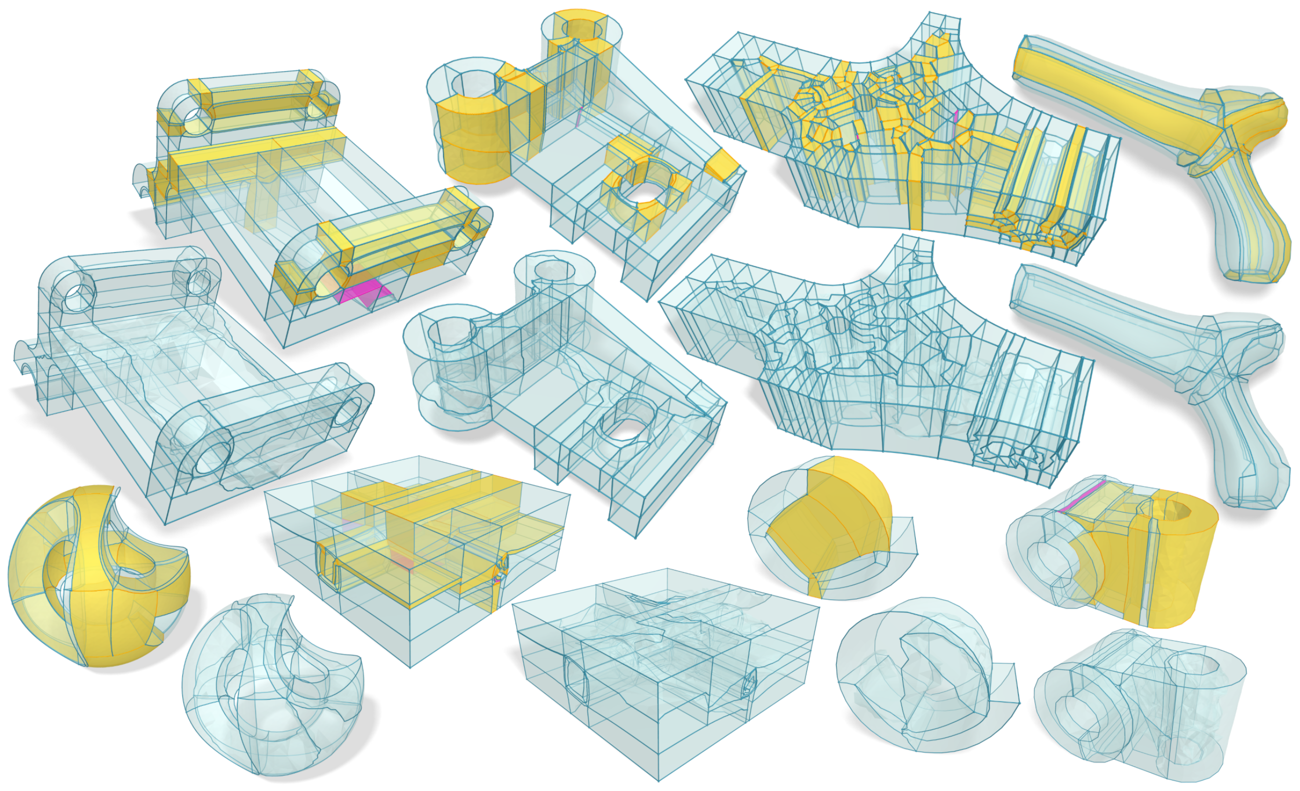


Fig. 15. Several models with embedded T-mesh complex, before and after performing our collapsing routine. Collapsing was performed in accordance to the maximally coarse quantization, eliminating all zero-arcs (orange), zero-patches (purple, if isolated), and zero-blocks (yellow) in the process. The embeddings of arcs and patches affected in the process were geometrically relaxed in the end (Sec. 6.2.2). Video animations of the process are available in the supplement.

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.

Fig. 15 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. 21, 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 research implementation based on a data structure not well 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 App. A.3) 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.

Mapping. 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. The block boundary maps were computed per patch as 2D Tutte embeddings. The interior map was initialized as a 3D Tutte embedding. For 99.96% of all blocks, the resulting map ϕ_b was fully bijective. The remaining 0.04% (4 blocks total from two different input models) have a few remaining inversions each with this strategy. Applying the reliable method of [Hinderink and Campen 2023], also for these a bijective map is obtained.

8.3 Comparison

Approaches for IGM recomputation (for a given quantization) using global non-convex optimization have recently been mentioned and

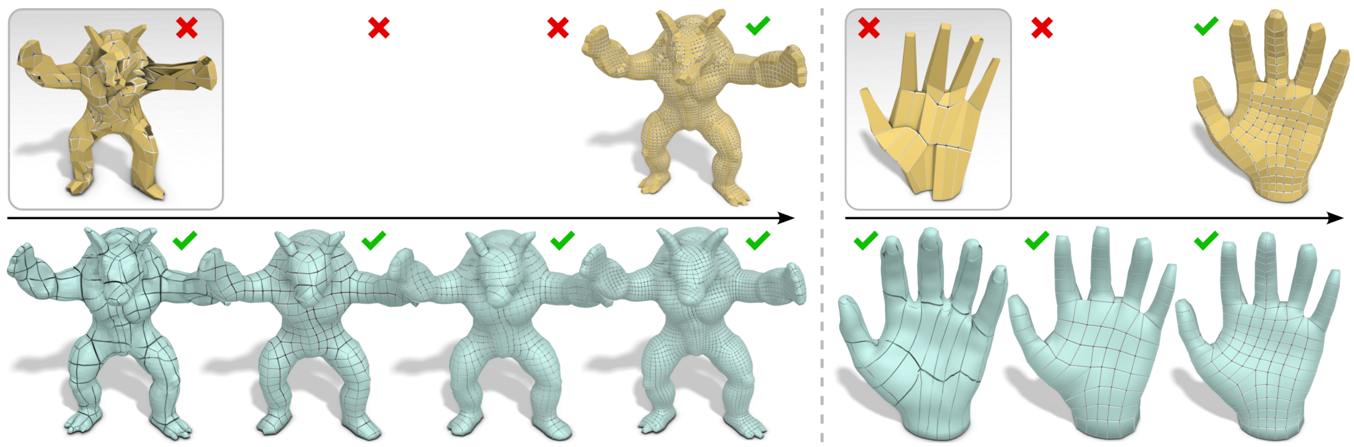


Fig. 16. Two examples of input instances (ARMADILLO and CAMILLE_HAND) demonstrating the unconditional robustness of our method (bottom row) in comparison to QGP3D-OPT (top row). The coarsest versions are shown with non-linear curved elements for visual clarity. QGP3D-OPT fails in obtaining a valid IGM 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.

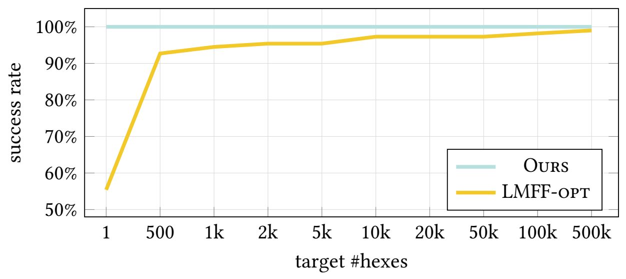


Fig. 17. To highlight the significance of robustness in IGM reparametrization, the failure rates of a state-of-the-art global reparametrization scheme LMFF-OPT applied to the 110 inputs from the third dataset are evaluated as a function of target complexity. Note that ‘target #hexes’ is what the quantization aims for; what ultimately results is of course limited by the prescribed singularity structure.

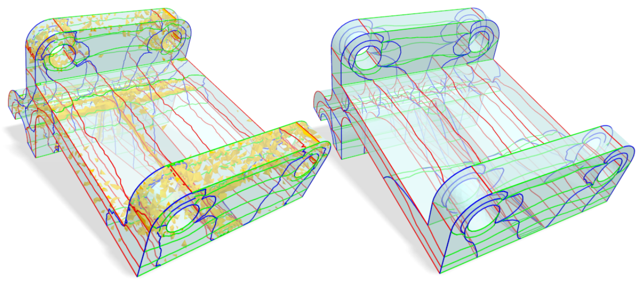


Fig. 18. Example (I06U M6) from the third dataset for which a previous approach, LMFF-OPT, yields a map (left) that inverts many tetrahedra (highlighted in orange) for a maximally coarse quantization setting. Our method obtains a valid IGM (right). Shown are the parametric integer-grid isocurves, colored according to local U- (red), V- (green) or W-alignment (blue).

used by Brückler et al. [2022a, Sec. 7.2] (based on [Jiang et al. 2014]) and by [Liu and Bommes 2023, Sec. 6.2], using somewhat different combinations of objectives, constraints, and solvers. We will call these QGP3D-OPT and LMFF-OPT, respectively, in the following—referring to just this optimization part, not the methods as a whole (focusing on quantization and frame field generation).

QGP3D-OPT. Brückler et al. [2022a] mention that they can reliably compute quantizations of arbitrary target resolution, but the subsequent computation of a corresponding IGM is non-robust especially for coarse settings. Our method resolves this issue. As an example, for the first two models from Table 1, as shown in Fig. 16, using our method we are able to compute valid IGMs for maximally coarse quantizations (as reported in Sec. 8.2), whereas QGP3D-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 22x and 10x 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: Fig. 17 shows how the success rate of LMFF-OPT decreases with a decreasing target mesh resolution. For a maximally coarse quantization, it fails in obtaining a valid IGM in 44% of the cases. Fig. 18 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 our method.

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. Our method should therefore not be misunderstood as aiming for a different mesh of potentially higher quality. Instead, the key benefit is the unconditional robustness our method 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. Table 4 reports 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.

8.4 Feature Alignment

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. In Table 3 the number of marked mesh elements per instance is reported.

Because we have extended the various stages of the motorcycle complex 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. Fig. 19 demonstrates this on an example from the dataset.

8.5 Block Structuring

Coarse IGMs are not only of academic interest (e.g. for stress tests as in Sec. 8.2). While maximal coarseness can imply excessive distortion (Fig. 21), 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 [Armstrong et al. 2015; Kopriva 2009]. This is similar to how *quad layouts* are used for the generation of patch-structured quad meshes [Lyon et al. 2021a; Campen 2017].

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. 20). Finally, a hex mesh can be

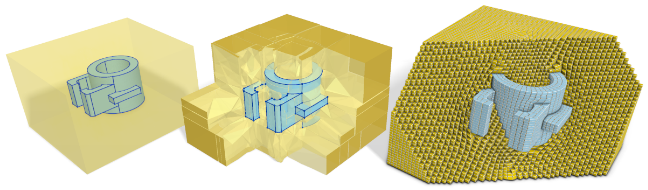


Fig. 19. Some input instances (like I15B S8, left) contain marked feature points, curves (dark blue), or surfaces (light blue) in the volume, that the result is supposed to respect. Our modified motorcycle complex construction routine properly aligns the T-mesh complex to these. Preserving feature nodes, 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.

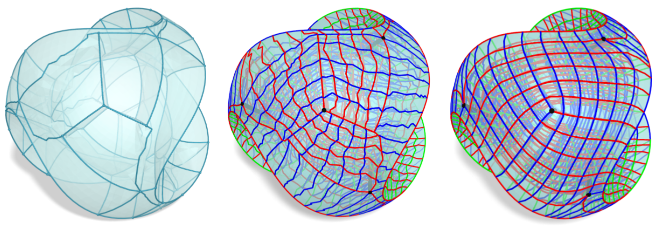


Fig. 20. While collapsing may result in some blocks being geometrically distorted or jagged (left), causing an initially distorted IGM after blockwise construction (center), a subsequent global map optimization starting from this valid initialization reduces map distortion (right)

extracted. Fig. 22 shows example results of this process; it visualizes the meshes' *base complex*, the coarsest conforming block structure that the meshes exhibit.

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 CONCLUSION AND FUTURE WORK

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 very 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.

Looking at the broader scope, the entire hexhahedral mesh generation pipeline outlined in Sec. 7.1, we note that step (1) (the computation of a global seamless map, possibly guided by a globally meshable frame field) demands further attention. Existing methods for this step do not yet offer full success guarantees, though recent progress can be observed [Liu and Bommes 2023]. This therefore is, and should be, a topic of ongoing research.

The ability to optimize singularity positions, similar to recent work on the 2D surface setting [Lyon et al. 2021b], would be of interest for improved quality (cf. Fig. 21) especially for coarse layouts, e.g. for improved block-structured mesh generation capabilities.

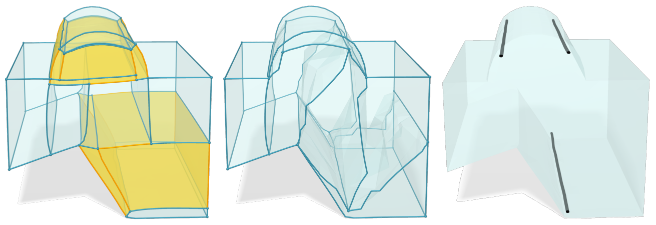


Fig. 21. Example of a block structure (left: input, middle: collapsed) resulting from a maximally coarse quantization, with low geometric quality—in particular due to fixed singularity positions (right).

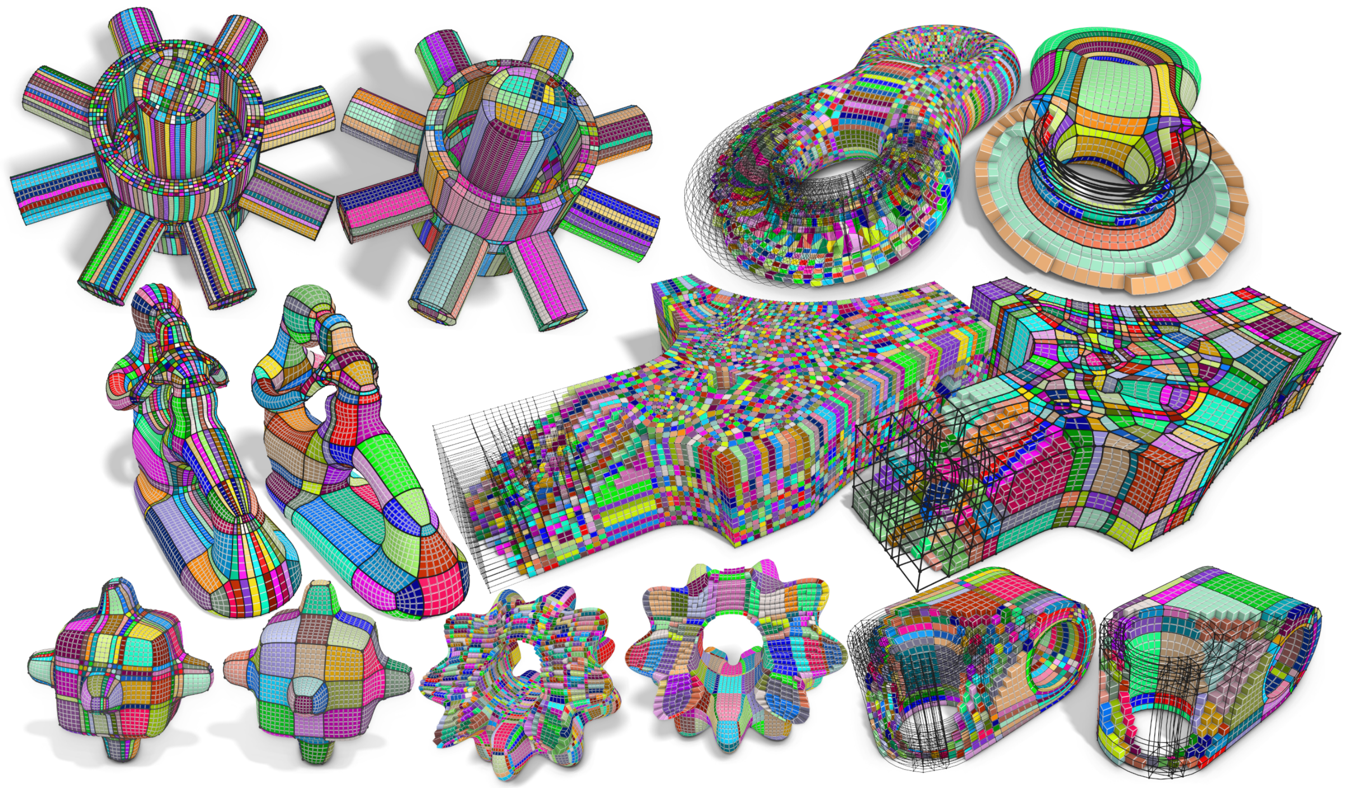


Fig. 22. Pairs of hexahedral meshes (of comparable resolution) with hexahedra colored according to blocks of the mesh's base complex. Some are sliced to reveal the interior. The right mesh of each pair (with a significantly simpler block structure) is the result obtained when imposing a coarse block layout as described in Sec. 8.5. 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. Models are number 1-5, 9 and 12 from the second dataset as listed in Table 2.

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. 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.

Finally, 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.

ACKNOWLEDGMENTS

The authors wish to thank David Bommes for inspiring discussions. This work was funded by the Deutsche Forschungsgemeinschaft (DFG) - 427469366; 456666331.

REFERENCES

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

P-A Beaufort, Maxence Reberol, Denis Kalmykov, Heng Liu, Franck Ledoux, and David Bommes. 2022. Hex Me If You Can. 41, 5 (2022), 125–134.

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013. Integer-Grid Maps for Reliable Quad Meshing. *ACM Trans. Graph.* 32, 4 (2013), 98:1–98:12.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph.* 28, 3 (2009), 77:1–77:10.

Janis Born, Patrick Schmidt, and Leif Kobbelt. 2021. Layout embedding via combinatorial optimization. In *Comp. Graph. Forum*, Vol. 40. 277–290.

Mario Botsch and Leif Kobbelt. 2004. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 185–192.

Michael L. Brewer, Lori A. Diachin, Patrick M. Knupp, Thomas Leurent, and Darryl J. Melander. 2003. The Mesquite Mesh Quality Improvement Toolkit. In *International Meshing Roundtable Conference.*

Heinz Bruggesser and Peter Mani. 1971. Shellable decompositions of cells and spheres. *Math. Scand.* 29, 2 (1971), 197–205.

Hendrik Brückler, David Bommes, and Marcel Campen. 2022a. Volume Parametrization Quantization for Hexahedral Meshing. *ACM Trans. Graph.* 41, 4 (2022).

Hendrik Brückler, Ojaswi Gupta, Manish Mandad, and Marcel Campen. 2022b. The 3D Motorcycle Complex for Structured Volume Decomposition. *Comp. Graph. Forum* 41, 2 (2022).

Marcel Campen. 2017. Partitioning surfaces into quadrilateral patches: A survey. *Comp. Graph. Forum* 36, 8 (2017), 567–588.

Marcel Campen, David Bommes, and Leif Kobbelt. 2015. Quantized global parametrization. *ACM Trans. Graph.* 34, 6 (2015).

Marcel Campen and Leif Kobbelt. 2014. Quad layout embedding via aligned parameterization. *Comp. Graph. Forum* 33, 8 (2014), 69–81.

Marcel Campen, Hanxiao Shen, Jiaran Zhou, and Denis Zorin. 2019. Seamless Parametrization with Arbitrary Cones for Arbitrary Genus. *ACM Trans. Graph.* 39, 1 (2019).

Marcel Campen, Cláudio T Silva, and Denis Zorin. 2016. Bijective maps from simplicial foliations. *ACM Trans. Graph.* 35, 4 (2016).

Gianmarco Cherchi, Marco Livesu, and Riccardo Scateni. 2016. Polycube simplification for coarse layouts of surfaces and volumes. *Comp. Graph. Forum* 35, 5 (2016), 11–20.

Etienne Corman and Keenan Crane. 2019. Symmetric Moving Frames. *ACM Trans. Graph.* 38, 4 (2019).

Joel Daniels, Claudio T Silva, and Elaine Cohen. 2009. Localized quadrilateral coarsening. 28, 5 (2009), 1437–1444.

Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Danny M. Kaufman, and Tao Ju. 2020. Lifting Simplices to Find Injectivity. *ACM Trans. Graph.* 39, 4 (2020).

Matthias Eck and Hugues 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.

David Eppstein, Michael T Goodrich, Ethan Kim, and Rasmus Tamstorf. 2008. Motorcycle graphs: canonical quad mesh partitioning. *Comp. Graph. Forum* 27, 5 (2008), 1477–1486.

Lori A Freitag and Carl Ollivier-Gooch. 1997. Tetrahedral mesh improvement using swapping and smoothing. *Internat. J. Numer. Methods Engrg.* 40, 21 (1997), 3979–4002.

Xifeng Gao, Zhigang Deng, and Guoning Chen. 2015. Hexahedral mesh re-parameterization from aligned base-complex. *ACM Trans. Graph.* 34, 4 (2015).

Xifeng Gao, Daniele Panozzo, Wenping Wang, Zhigang Deng, and Guoning Chen. 2017. Robust Structure Simplification for Hex Re-Meshing. *ACM Trans. Graph.* 36, 6 (2017).

Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. 2021. Foldover-Free Maps in 50 Lines of Code. *ACM Trans. Graph.* 40, 4 (2021).

Leo Grady. 2008. Minimal surfaces extend shortest path segmentation methods to 3D. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 2 (2008), 321–334.

Erkan Gunpinar, Marco Livesu, and Marco Attene. 2023. Exploration of 3D motorcycle complexes from hexahedral meshes. *Computers & Graphics* (2023).

Steffen Hinderink and Marcel Campen. 2023. Galaxy Maps: Localized Foliations for Bijective Volumetric Mapping. *ACM Trans. Graph.* 42, 4 (2023).

Tengfei Jiang, Jin Huang, Yuanzhen Wang, Yiying Tong, and Hujun Bao. 2014. Frame field singularity correction for automatic hexahedralization. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (2014), 1189–1199.

Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover - Surface Parameterization using Branched Coverings. *Comp. Graph. Forum* 26, 3 (2007), 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.

Reinhard Klette. 2000. Cell complexes through time. In *Vision Geometry IX*, Longin Jan Latecki, David M. Mount, and Angela Y. Wu (Eds.), Vol. 4117. International Society for Optics and Photonics, SPIE, 134 – 145.

David A. Kopriva. 2009. *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*. Springer Netherlands, Chapter Spectral Element Methods, 293–354.

Vladislav Kraevoy and Alla Sheffer. 2004. Cross-parameterization and compatible remeshing of 3D models. *ACM Trans. Graph.* 23, 3 (2004), 861–869.

Vladislav Kraevoy, Alla Sheffer, and Craig Gotsman. 2003. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph.* 22, 3 (2003), 326–333.

Michael Kremer, David Bommes, and Leif Kobbelt. 2013. OpenVolumeMesh–A versatile index-based data structure for 3D polytopal complexes. In *Proceedings of the 21st International Meshing Roundtable*. 531–548.

Zohar Levi. 2021. Direct Seamless Parametrization. *ACM Trans. Graph.* 40, 1 (2021).

Zohar Levi. 2022. Seamless Parametrization of Spheres with Controlled Singularities. *Comp. Graph. Forum* (2022).

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-Hex Meshing Using Singularity-Restricted Field. *ACM Trans. Graph.* 31, 6 (2012).

Juncong Lin, Xiaogang Jin, Zhengwen Fan, and Charlie CL Wang. 2008. Automatic polycube-maps. In *Int. Conference on Geometric Modeling and Processing*. 3–16.

Heng Liu and David Bommes. 2023. Locally Meshable Frame Fields. *ACM Transactions on Graphics* 42, 4 (2023).

Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* 37, 4 (2018).

Marco Livesu, Nico Pietroni, Enrico Puppo, Alla Sheffer, and Paolo Cignoni. 2020. LoopyCuts: Practical Feature-Preserving Block Decomposition for Strongly Hex-Dominant Meshing. *ACM Trans. Graph.* 39, 4 (2020).

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. Polycut: Monotone graph-cuts for polycube base-complex construction. *ACM Trans. Graph.* 32, 6 (2013).

Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx: Robust Hexahedral Mesh Extraction. *ACM Transactions on Graphics* 35, 4 (2016).

Max Lyon, Marcel Campen, David Bommes, and Leif Kobbelt. 2019. Parametrization Quantization with Free Boundaries for Trimmed Quad Meshing. *ACM Trans. Graph.* 38, 4 (2019).

Max Lyon, Marcel Campen, and Leif Kobbelt. 2021a. Quad Layouts via Constrained T-Mesh Quantization. *Comp. Graph. Forum* 40, 2 (2021).

Max Lyon, Marcel Campen, and Leif Kobbelt. 2021b. Simpler Quad Layouts using Relaxed Singularities. *Comp. Graph. Forum* 40, 5 (2021), 169–179.

Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust field-aligned global parametrization. *ACM Trans. Graph.* 33, 4 (2014).

Ashish Myles and Denis Zorin. 2012. Global parametrization by incremental flattening. *ACM Trans. Graph.* 31, 4 (2012), 109:1–109:11.

M. Nieser, U. Reitebuch, and K. Polthier. 2011. CubeCover – Parameterization of 3D Volumes. *Comp. Graph. Forum* 30, 5 (2011), 1397–1406.

Valentin Z. Nigolian, Marcel Campen, and David Bommes. 2023. Expansion Cones: A Progressive Volumetric Mapping Framework. *ACM Trans. Graph.* 42, 4 (2023).

Stefano Nuvoli, Alex Hernandez, Claudio Esperança, Riccardo Scateni, Paolo Cignoni, and Nico Pietroni. 2019. QuadMixer: layout preserving blending of quadrilateral meshes. *ACM Trans. Graph.* 38, 6 (2019).

David Palmer, David Bommes, and Justin Solomon. 2020. Algebraic Representations for Volumetric Frame Fields. *ACM Trans. Graph.* 39, 2 (2020).

Chi-Han Peng, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2011. Connectivity editing for quadrilateral meshes. In *Proceedings of the 2011 SIGGRAPH Asia conference*.

Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean Remacle, and Marco Livesu. 2022. Hex-Mesh Generation and Processing: A Survey. *ACM Trans. Graph.* 42, 2 (2022).

Nico Pietroni, Stefano Nuvoli, Thomas Alderighi, Paolo Cignoni, Marco Tarini, et al. 2021. Reliable feature-line driven quad-remeshing. *ACM Transactions on Graphics* 40, 4 (2021).

Emil Praun, Wim Sweldens, and Peter Schröder. 2001. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 179–184.

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 4 (2017).

Maxence Reberol, Alexandre Chemin, and Jean-Francois Remacle. 2019. Multiple Approaches to Frame Field Correction for CAD Models. In *Proc. 28th International Meshing Roundtable*.

Patrick Schmidt, Janis Born, Marcel Campen, and Leif Kobbelt. 2019. Distortion-minimizing injective maps between surfaces. *ACM Trans. Graph.* 38, 6 (2019).

Patrick Schmidt, Marcel Campen, Janis Born, and Leif Kobbelt. 2020. Inter-surface maps via constant-curvature metrics. *ACM Trans. Graph.* 39, 4 (2020), 119–1.

John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. 2004. Inter-surface mapping. In *ACM SIGGRAPH 2004 Papers*. 870–877.

Chun Shen, Shuming Gao, and Rui Wang. 2021. Topological operations for editing the singularity on a hex mesh. *Engineering with Computers* 37, 2 (2021), 1357–1375.

Hanxiao Shen, Leyi Zhu, Ryan Capouellez, Daniele Panozzo, Marcel Campen, and Denis Zorin. 2022. Which cross fields can be quadrangulated? Global parameterization from prescribed holonomy signatures. *ACM Trans. Graph.* 41, 4 (2022).

Kenshi Takayama. 2019. Dual Sheet Meshing: An Interactive Approach to Robust Hexahedralization. *Comp. Graph. Forum* 38, 2 (2019), 37–48.

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. Polycube-maps. *ACM Trans. Graph.* 23, 3 (2004), 853–860.

Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing (SGP '06)*. Eurographics Association, 201–210.

Isaac J Trotts, Bernd Hamann, Kenneth I Joy, and David F Wiley. 1998. *Simplification of tetrahedral meshes*. IEEE.

W. T. Tutte. 1963. How to draw a graph. *Proc. Lond. Math. Soc.* 13 (1963), 743–767.

Ryan Viertel, Matthew L Staten, and Franck Ledoux. 2016. *Analysis of Non-Meshable Automatically Generated Frame Fields*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Ofir Weber and Denis Zorin. 2014. Locally injective parametrization with arbitrary fixed boundaries. *ACM Trans. Graph.* 33, 4 (2014).

Jiaran Zhou, Changhe Tu, Denis Zorin, and Marcel Campen. 2020. Combinatorial construction of seamless parameter domains. *Comp. Graph. Forum* 39, 2 (2020), 179–190.

A ALGORITHM DETAILS

Alg. 1 details the ArcPatchShift. The case differentiation in line 1 distinguishes whether arc a initially lies on one or two edges of the triangle f it is shifted across. 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 then erased from the arc afterwards (line **2**). Both variants are equivalent, but the latter allows for a more concise notation.

Algorithm 1: ArcPatchShift(a, f)

```
Input: Tet mesh $\mathcal{M} = \{C, F, E, V\}$, cell complex $\mathcal{S} = \{B, P, A, N\}$,
  embedding $\mathcal{I}_\mathcal{S} \triangleq \overleftarrow{\mathcal{I}}_\mathcal{M}$, arc $a \in A$, incident face $f \in F$
Output: Consistent $\mathcal{I}_\mathcal{S}$ with $a$ shifted across $f$
$E_a$ = edges $e$ of $f$ with $\overleftarrow{\mathcal{I}}_e = a$
ArcShift$(a, f)$
1 $x = \begin{cases} e \in E_a & \textbf{if} |E_a| = 1 \\ v \in V \text{ shared by } e_1, e_2 \in E_a & \textbf{otherwise} \end{cases}$
repeat
    $B_f$ = blocks incident to $f$
    foreach block $b \in B_f$ do
        while there is a patch $p$ of $b$ incident to $x$ do
            while there is a face $f' \in \mathcal{I}_p$ incident to $x$ do
                $c$ = cell of block $b$ incident to $f'$
                PatchShift$(p, c)$
2               $\overleftarrow{\mathcal{I}}_f = 0$
until no embedding updates performed
```

Algorithm 2: NodeArcPatchShift(n, e)

```
Input: Tet mesh $\mathcal{M} = \{C, F, E, V\}$, cell complex $\mathcal{S} = \{B, P, A, N\}$,
  embedding $\mathcal{I}_\mathcal{S} \triangleq \overleftarrow{\mathcal{I}}_\mathcal{M}$, node $n \in N$, edge $e \in E$
Output: consistent embedding $\mathcal{I}_\mathcal{S}$ with $n$ shifted across $e$
$v = \mathcal{I}_n$
NodeShift$(n, e)$
repeat
    $P_e$ = patches incident to $e$
    $B_e$ = blocks incident $e$
    foreach patch $p \in P_e$ do
        if there is an arc $a$ of $p$ incident to $v$ then
            $a$ = arc of $p$ incident to $v$
            while there is an edge $e \in \mathcal{I}_a$ incident to $v$ do
                $f$ = face $f \in \mathcal{I}_p$ incident to $e$
                ArcPatchShift$(a, f)$
1               $\overleftarrow{\mathcal{I}}_e = 0$
    foreach block $b \in B_e$ do
        while there is a patch $p$ of $b$ incident to $v$ do
            while there is a face $f \in \mathcal{I}_p$ incident to $v$ do
                $c$ = cell $c \in \mathcal{I}_b$ incident to $f$
                PatchShift$(p, c)$
until no embedding updates performed
```

Alg. 2 details ArcPatchShift. 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.

Note that both, ArcPatchShift and NodeArcPatchShift, contain unordered loops over incident elements. These can indeed be processed in arbitrary order, this has only geometric effects on the resulting embedding.

Alg. 3 details the execution of the patch bisection operator and Alg. 4 that of the block bisection operator, assuming executability.

Algorithm 3: BisectPatch(p)

```
Input: Aligned T-mesh $\mathcal{T} = \{B, P, A, N\}$, quantization $\ell_A$,
  embedding $\overleftarrow{\mathcal{I}}_\mathcal{M}$, quasi-pillow patch $p = \{a_1, a_2, \ldots, a_n\}$
Output: $p$ bisected
$[S_\uparrow, S_\downarrow]$ = any opposite sides with a total of three or more arcs
$[a_\uparrow, a_\downarrow]$ = first arcs of $S_\uparrow, S_\downarrow$ on same end of $p$, s.t. $\ell_{a_\uparrow} \geqslant \ell_{a_\downarrow}$
if $\ell_{a_\uparrow} > \ell_{a_\downarrow}$ then
    $v$ = choose vertex in interior of $\mathcal{I}_a$
    add node $n_\text{new}$ with $\mathcal{I}_{n_\text{new}} = v$
    split $a_\uparrow = (n_0, n_1)$ into $a_1 = (n_0, n_\text{new})$ and $a_2 = (n_\text{new}, n_1)$
    split $\mathcal{I}_{a_\uparrow}$ into $\mathcal{I}_{a_1}$ and $\mathcal{I}_{a_2}$ at $v$
    $\ell_{a_1} = \ell_{a_\downarrow}$ and $\ell_{a_2} = \ell_{a_\uparrow} - \ell_{a_\downarrow}$
    $a_\uparrow = \begin{cases} a_1 & \text{if } a_1 \text{ is on same end of } p \text{ as } a_\downarrow, \\ a_2 & \text{otherwise} \end{cases}$
$[n_\uparrow, n_\downarrow]$ = second nodes of $a_\uparrow, a_\downarrow$
$[A_\leftarrow, A_\rightarrow]$ = partition arcs of $p$ at $n_\uparrow, n_\downarrow$
$E_\updownarrow$ = shortest path between $\mathcal{I}_{n_\uparrow}$ and $\mathcal{I}_{n_\downarrow}$ through interior of $\mathcal{I}_p$
$a_\text{new}$ = new arc between $n_\uparrow$ and $n_\downarrow$
$\mathcal{I}_{a_\text{new}} = E_\updownarrow$ and $\ell_{a_\text{new}} = 0$
split $p$ into $p_\leftarrow = A_\leftarrow \cup \{a_\text{new}\}$ and $p_\rightarrow = A_\rightarrow \cup \{a_\text{new}\}$
split $\mathcal{I}_p$ into $\mathcal{I}_{p_\leftarrow}$ and $\mathcal{I}_{p_\rightarrow}$ at $\mathcal{I}_{a_\text{new}}$
```

Algorithm 4: BisectBlock(b)

```
Input: Aligned T-mesh $\mathcal{T} = \{B, P, A, N\}$, quantization $\ell_A$,
  embedding $\overleftarrow{\mathcal{I}}_\mathcal{M}$, quasi-pillow block $b = \{p_1, p_2, \ldots, p_n\}$
Output: $b$ bisected
while there is a patch $p$ with more than 4 arcs on $b$ do
    BisectPatch$(p)$
$A^\circ$ = any cycle of $\geqslant 2$ equally aligned arcs on block $b$
$[P_\leftarrow, P_\rightarrow]$ = partition patches of $b$ at $A^\circ$
$p_\text{new}$ = new patch incident to $A^\circ$
$\mathcal{I}_{p_\text{new}} = \bigcup_{p' \in P_\leftarrow} \mathcal{I}_{p'}$
while $\mathcal{I}_{p_{new}}$ overlaps with any $\mathcal{I}_{p'}$ do
    foreach cell $c \in \mathcal{I}_b$ incident to the overlap do
        PatchShift$(p_\text{new}, c)$
split $b$ into $b_\leftarrow = P_\leftarrow \cup \{p_\text{new}\}$ and $b_\rightarrow = P_\rightarrow \cup \{p_\text{new}\}$
split $\mathcal{I}_b$ into $\mathcal{I}_{b_\leftarrow}$ and $\mathcal{I}_{b_\rightarrow}$ at $\mathcal{I}_{p_\text{new}}$
```

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.

A.1 Boundaries

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. 9p-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.

Boundary Elements. Cell complex boundary elements are (in the usual case of a surjective embedding) embedded in the background mesh's boundary. To maintain surjectivity, they need to be constrained to stay within that boundary. For that reason a boundary node is only allowed to be shifted across a boundary edge and a boundary arc is only allowed to be shifted across a boundary face. When an arc is shifted across a facet within the boundary, note that the boundary patch dragged behind it is necessarily the last successor and thus, by the above skip rule, simply is extended by the traversed boundary face. It thus remains on the boundary.

For a node shift within the boundary additional care needs to be taken, as the order in which the incident arcs are processed is not uniquely determined. To keep an incident boundary arc embedded in the boundary, it must be picked in the loop of Alg. 2 only when it is reached via a boundary patch. It may be reached (earlier) via a non-boundary patch, but shifting it via such a patch would lift it off the boundary.

In general, because the collapse operators are *directed*, i.e. they move one side of the collapsed element onto the other side, in a collapse where one side is on the boundary, this side must always be the stationary side. A collapse of a non-boundary element whose sides are both on the boundary cannot be performed consistently.

A.2 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.

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 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. 23). Then recompute the fan.

II 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. Then recompute the fan.

III Before an 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. 24).

IV Before an 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. 25).

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.

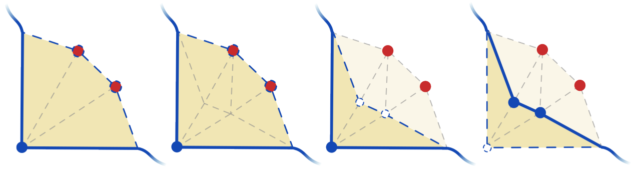


Fig. 23. Background mesh refinement of a triangle fan (or a tet fan in cross section view) to prevent overlaps with forbidden vertices (red) during NodeArcPatchShift or ArcPatchShift. In the end, shifts can be carried out over subtriangles (or subtetrahedra) without causing overlaps.

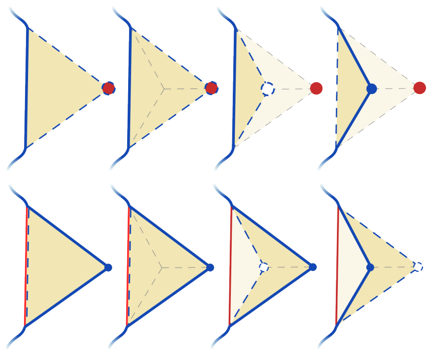


Fig. 24. Background mesh refinement of a triangle (1:3 split) to prevent overlap with a forbidden vertex or edge (red) in ArcShift.

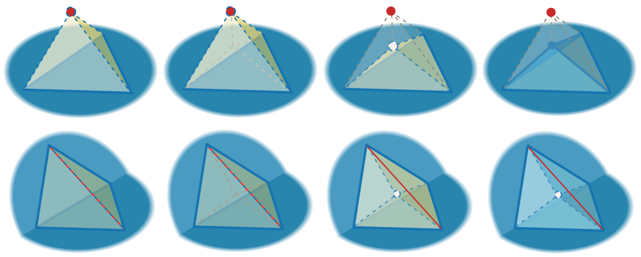


Fig. 25. Background mesh refinement of a tetrahedron (1:4 split) to prevent overlap with a forbidden vertex or edge (red) in PatchShift.

A.3 Background Mesh Remeshing

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. For simplicity we make use of a straightforward greedy approach: The standard operations edge collapse, edge split, face swap, and vertex shift are greedily applied on the background mesh wherever this locally improves the worst inner (facet and tet) angles. 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.

B FEATURE SUPPORT

We will consider in the following three types of features that the to be generated hexahedral mesh may be asked to align to: feature points, feature curves, and feature surfaces. We assume that such points coincide with vertices of the background mesh, while curves and surfaces coincide with edges or faces, respectively. The mesh elements associated to features will consequently be referred to as feature vertices, edges, and facets. They may also coincide with boundaries of the mesh or singularities of a seamless map on the mesh. In fact, our approach to extend the motorcycle complex generation method of Brückler et al. [2022b] and the quantization method of Brückler et al. [2022a] to support such features is based on handling feature curves similarly to singular curves and feature surfaces similarly to boundary regions.

Seamless Parametrization. Consider first the process of generating a seamless parametrization (or *map*) for a given background mesh and an associated singularity graph, step (1) of the pipeline outlined in Sec. 7.1, as described in detail for instance in [Nieser et al. 2011; Liu et al. 2018]. The usual requirement (imposed by constraints in this process) is that both, boundary facets and singular edges, must be *aligned* under the parametrization, meaning they coincide with (part of) an iso-surface or iso-curve of the parametrization. This results in (irregular) edges and boundary facets of the later to be generated hex mesh geometrically matching singularities and boundaries of the object. The same should be achieved for feature curves and surfaces—as well as for feature points, which hex mesh vertices should coincide with. In the following we therefore assume that features, in analogy to singularities, have been taken into account by means of alignment constraints in the process of generating the seamless parametrization.

To simplify handling the features, we assume these conventions are followed by the set of features:

- The end points of a feature curve are marked as feature points (or the curve is cyclic).
- The boundary of a feature surface is marked as feature curves (or the surface forms a closed manifold surface).
- The intersection of any two differently aligned feature curves is marked as feature point.
- The intersection of any two differently aligned feature surfaces is marked as feature curve and has constant alignment.

Numerical Sanitization. To then safely facilitate the construction of the motorcycle complex based on the generated volumetric seamless parametrization, the numerical sanitization method described in [Brückler et al. 2022b] can be adapted to respect feature constraints by two key modifications:

- Treat feature facets equivalently to boundary facets for the purpose of determining cut edges and align sheets.
- Introduce align edges (isolated feature curves not incident to any align sheet) and align branches, i.e. connected sets of align edges bounded by additional (or existing) nodes. Additional alignment constraints that not only constrain one but two coordinate components are introduced for the end nodes of each align branch. Back-substititution for non-node vertices can be performed for such align branches exactly as is done for singular branches.

Feature-Aligned T-Mesh. Next, to guarantee representation of features in the resulting hexahedral mesh we have to make sure these get represented in the intermediate T-mesh. To ensure this, we augment the construction algorithm for the motorcycle complex [Brückler et al. 2022b] based on the feature-aligned seamless parametrization by the following rules:

- Mark all feature facets as walls.
- Treat non-singular feature edges like singular edges, i.e. spawn four expanding walls on them, each expanding in one of the parametric directions orthogonal to the aligned edge.
- At each feature node, for all three parametric dimensions spawn a wall.
- In the complex reduction routine preserve all walls that are incident on features.

Feature-Separating Quantization. Finally, during quantization computation [Brückler et al. 2022a], to prevent features from getting collapsed onto each other under the quantization we treat them similarly to singularities and boundaries (which likewise must not be collapsed onto each other). To this end, we define three additional critical entity classes in accordance with [Brückler et al. 2022a]:

Definition 1 (Feature Link). *A maximal sequence of one or more feature arcs connected via regular nodes is a feature link. Note that it may connect two feature nodes, form a loop rooted at one feature node, or form a cycle.*

Definition 2 (Feature Region). *A maximal set of one or more feature patches connected via regular arcs is a feature region. Note that it can be a closed surface or be bounded by feature arcs.*

The set of *critical entities*, that the method relies on, is then extended to include these two classes, as well as the feature nodes. When then executing the quantization procedure, it produces a quantization on the feature-aligned T-mesh that properly separates boundaries, singularities, *and* features.

Table 2. For inputs from the second 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, arc) and bisections (block, patch) performed to remove all zero-elements are shown in grey, followed by the grand total of operations. The last column reports the relative decrease in T-mesh complexity.

Model	$\lvert B_0\rvert$		$\lvert P_0\rvert$		$\lvert A_0\rvert$		$\frac{\lvert\mathcal{S}_0\rvert}{\lvert\mathcal{S}\rvert}$	$\updownarrow_B$	$\updownarrow_P$	$\updownarrow_A$	$\vert_B$	$\vert_P$	$\sum$	$\Delta\lvert\mathcal{S}\rvert$
Fcbpsvdfhm Example 2	149	51%	625	45%	636	30%	37%	172	654	617	23	52	1518	58%
AMGvVPD bumpy torus	91	19%	486	23%	497	18%	20%	115	529	481	24	49	1198	33%
Agatmsmg Example 3	71	20%	309	20%	310	14%	17%	88	345	314	17	42	806	28%
Psfclosav cubespikes model out	34	20%	242	29%	270	22%	25%	71	305	279	33	71	759	39%
AMUCIP hollow-eight-hex	24	21%	117	22%	125	16%	19%	80	228	173	56	123	660	34%
Psfclosav cubespikes model in	26	17%	209	28%	242	23%	24%	47	239	230	17	34	567	40%
Agatmsmg Example 2	34	14%	159	14%	156	10%	12%	62	218	186	28	62	556	26%
Agatmsmg Example 4	41	15%	181	15%	165	10%	12%	65	229	188	24	48	554	25%
SPfPHM Double hinge WH	32	54%	136	48%	132	31%	39%	64	200	160	32	84	540	59%
PHOvER impeller stresstest out	26	17%	137	17%	148	13%	15%	60	200	177	34	48	519	30%
AMUCIP rockerarm-hex	14	7%	96	11%	104	9%	10%	37	141	123	23	45	369	18%
EVPC Fertility hex-largel	34	14%	131	14%	114	9%	11%	46	155	124	12	21	358	21%
Psfclosav rockerarm polycube in	9	18%	43	17%	43	11%	14%	37	101	71	28	72	309	24%
Psfclosav cubespikes polycube out	23	46%	138	51%	158	37%	42%	28	132	125	2	2	289	60%
Psfclosav cubespikes polycube in	24	49%	131	49%	151	36%	41%	28	128	123	3	0	282	59%
ECMfGTS femur shell1	25	60%	102	52%	104	35%	43%	29	113	110	4	12	268	67%
Psfclosav rockerarm polycube out	16	29%	73	26%	71	17%	21%	26	92	78	10	21	227	33%
PMGfPBC bunny hex opt	15	12%	69	13%	66	10%	11%	25	91	74	10	25	225	19%
Psfclosav chinese dragon polycube in	12	24%	67	25%	71	17%	21%	24	89	76	11	24	224	36%
PMGfPBC BU hex opt	17	13%	69	13%	62	9%	11%	24	80	64	7	11	186	18%
AMuSF rod	9	26%	43	25%	44	17%	21%	20	68	58	11	27	184	32%
AMuSF double	7	14%	42	19%	41	14%	16%	24	67	49	17	25	182	29%
SPfPHM Double hinge NH	8	19%	38	18%	44	14%	16%	16	62	60	8	24	170	28%
AMGvVPD asm001	5	6%	32	9%	30	6%	7%	21	58	40	16	30	165	12%
SPfPHM Gear	19	39%	70	29%	64	17%	23%	19	70	64	0	0	153	37%
lCoPMfCS rod	9	10%	38	11%	32	7%	9%	19	58	42	10	22	151	14%
Psfclosav bunny polycube out	12	32%	56	29%	61	21%	25%	14	60	57	2	7	140	40%
SVDvGS twistedu	6	29%	34	35%	32	23%	28%	16	52	40	10	18	136	45%
SVDvGS rotellipse padded	6	29%	34	34%	32	23%	28%	16	52	40	10	17	135	45%
AMuSF joint	11	32%	44	27%	41	17%	22%	15	56	47	4	12	134	34%
HMGvCQ pone.0177603.s003	11	32%	44	27%	41	17%	22%	15	56	47	4	12	134	34%
Psfclosav bunny polycube in	11	31%	51	28%	55	19%	23%	14	57	54	3	5	133	39%
SPfPHM Wrench	11	31%	47	27%	50	19%	23%	13	53	54	2	6	128	38%
Psfclosav block polycube out	10	29%	47	28%	54	20%	24%	12	52	56	2	4	126	38%
AMuSF rockerarm	6	4%	46	8%	54	7%	7%	10	52	54	4	2	122	13%
AMUCIP fancy ring-hex	12	55%	50	49%	51	34%	41%	14	52	47	2	6	121	59%
DSM fandisk	8	24%	36	22%	37	15%	18%	13	46	42	5	10	116	32%
Psfclosav chinese dragon polycube out	9	20%	44	19%	46	13%	16%	11	49	46	2	7	115	24%
AMuSF sculpture-A	13	48%	52	41%	52	28%	34%	13	50	48	0	0	111	52%
SPfPHM Column	10	37%	46	36%	46	24%	30%	12	46	42	2	2	104	44%
Psfclosav block polycube in	8	24%	34	20%	36	14%	17%	10	40	40	2	6	98	30%
Psfclosav fandisk polycube in	4	20%	18	17%	18	11%	14%	11	34	26	7	18	96	21%
DSM fandisk.liu18	6	21%	27	20%	27	13%	16%	11	37	32	5	10	95	30%
AMuSF sculpture-B	10	30%	40	26%	43	18%	22%	10	38	39	0	0	87	32%
Psfclosav asm polycube in	1	5%	4	4%	4	3%	3%	9	24	16	8	24	81	1%
SPfPHM Chamfer L4	7	54%	27	43%	26	26%	34%	8	30	28	1	3	70	53%
Psfclosav teapot polycube in	1	4%	14	10%	20	8%	9%	6	23	23	5	12	69	16%
Psfclosav fandisk polycube out	5	22%	21	18%	20	11%	15%	7	28	24	2	7	68	27%
Psfclosav teapot polycube out	1	4%	11	7%	16	7%	7%	6	22	22	5	13	68	12%
Psfclosav asm polycube out	3	16%	12	13%	12	8%	10%	5	22	20	2	10	59	16%
AMuSF fandisk	5	20%	22	18%	22	12%	15%	6	24	23	1	2	56	27%
HMGvCQ fandisk	5	20%	21	18%	20	11%	14%	6	23	21	1	2	53	25%
HMGvCQ pone.0177603.s002	4	16%	18	15%	18	10%	12%	5	20	19	1	2	47	25%
Psfclosav hand polycube in	5	25%	18	18%	17	11%	14%	5	18	17	0	0	40	22%
Psfclosav hand polycube out	2	11%	9	9%	11	7%	8%	2	9	11	0	0	22	13%
AMUCIP knot-hex	3	8%	10	7%	8	4%	6%	3	10	8	0	0	21	9%
Psfclosav table2 polycube in	1	8%	5	8%	6	6%	6%	2	8	8	1	2	21	15%
AMUCIP joint-hex	2	10%	9	9%	8	5%	7%	2	9	8	0	0	19	11%
AMuSF ellipsoid-A	1	7%	6	9%	6	6%	7%	2	7	6	1	0	16	12%
Psfclosav femur polycube out	1	6%	6	7%	7	5%	6%	1	6	7	0	0	14	12%
lCoPMfCS kitty	1	2%	6	3%	6	2%	2%	1	6	6	0	0	13	4%
AMUCIP kitten-hex	1	2%	6	3%	6	2%	2%	1	6	6	0	0	13	4%
AMuSF hanger	1	4%	4	4%	4	2%	3%	1	4	4	0	0	9	5%
ECMfGTS femur shell0	1	11%	4	9%	4	6%	7%	1	4	4	0	0	9	11%
DSM hanger	1	5%	4	4%	4	3%	3%	1	4	4	0	0	9	5%
SMF hex brokenbullet	0	0%	2	3%	4	4%	3%	0	2	4	0	0	6	5%
Psfclosav table1 polycube in	0	0%	1	1%	2	1%	1%	0	1	2	0	0	3	2%
Psfclosav table1 polycube out	0	0%	1	1%	2	1%	1%	0	1	2	0	0	3	5%

Table 3. For inputs from the third dataset, numbers of marked feature elements (facets, edges, and vertices), as well as 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, arc) and bisections (block, patch) performed to remove all zero-elements are shown in grey, followed by the grand total of operations. The last column reports the relative decrease in T-mesh complexity.

Model	$\|F_{\text{feature}}\|$	$\|E_{\text{feature}}\|$	$\|V_{\text{feature}}\|$	$\|B_0\|$		$\|P_0\|$		$\|A_0\|$		$\frac{\|\mathcal{S}_0\|}{\|\mathcal{S}\|}$	$\updownarrow B$	$\updownarrow P$	$\updownarrow A$	$\|B$	$\|P$	Σ	$\Delta\|\mathcal{S}\|$
i18b s22	27680	1578	60	224	38%	931	34%	825	22%	28%	354	1185	906	123	299	2867	44%
i06b m6	23964	1545	76	254	43%	1112	42%	1033	29%	35%	321	1191	946	63	100	2621	55%
i13c s6	18872	1544	99	170	49%	675	41%	620	26%	33%	263	879	713	93	224	2172	54%
i13u s6	43268	2609	99	177	53%	699	44%	637	28%	36%	266	886	721	89	205	2167	57%
i15b s8	11798	944	71	138	29%	629	28%	576	19%	23%	246	835	648	106	233	2068	40%
i09c m9	74688	2440	76	97	43%	359	34%	321	21%	28%	210	621	465	113	281	1690	44%
i09u m9	81796	2496	76	93	41%	336	32%	293	19%	25%	209	614	454	116	291	1684	41%
s08b cross cyls dr	8188	480	14	122	48%	545	46%	505	31%	38%	168	614	487	44	76	1389	61%
i14b s7	9576	631	23	101	40%	421	35%	366	22%	28%	174	561	420	72	160	1387	45%
i01u m1	27178	1838	76	99	39%	411	35%	406	24%	29%	123	455	413	24	56	1071	47%
n05c box min pcyls	8852	654	27	41	17%	217	18%	220	13%	15%	121	372	287	80	182	1042	28%
i01c m1	14788	1179	76	102	43%	402	36%	385	24%	30%	125	446	396	23	47	1037	49%
n05u box min pcyls	17052	1032	27	42	18%	221	20%	220	14%	17%	113	360	279	68	166	986	29%
i23c s31	11486	1042	76	66	33%	278	29%	269	19%	24%	107	362	307	41	96	913	38%
i17u s20	26960	1421	51	77	37%	310	32%	288	21%	26%	104	367	313	27	63	874	44%
i12u s5	31836	2304	88	83	40%	313	33%	295	21%	27%	103	364	321	20	50	858	44%
i02u m2	35866	1842	46	69	42%	294	37%	290	25%	31%	99	354	303	30	52	838	51%
i12c s5	17486	1485	88	79	41%	299	34%	280	22%	28%	98	345	299	19	47	808	46%
i06u m6	36658	2285	68	69	53%	275	45%	262	29%	37%	93	329	280	23	61	786	56%
i06c m6	27010	1858	68	66	55%	257	45%	244	29%	37%	88	309	265	22	59	743	57%
i02c m2	14296	1030	46	66	40%	282	36%	284	25%	31%	85	311	277	18	35	726	49%
i22c s27	10618	1134	94	51	27%	228	25%	230	17%	21%	74	285	255	23	56	693	35%
i25u s40	25736	1971	77	47	34%	208	31%	193	19%	25%	80	261	201	31	69	642	41%
i18u s22	27850	1803	52	60	55%	222	43%	195	26%	34%	78	257	208	18	38	599	54%
i18c s22	11188	1000	52	60	53%	236	44%	216	27%	35%	73	257	216	13	26	585	56%
n08c pentapyr	2184	179	6	35	53%	150	44%	133	27%	35%	74	213	148	36	71	542	54%
n03b skijump box cyl	10926	621	20	35	21%	161	22%	146	15%	18%	64	210	159	29	64	526	29%
i20u s25	47926	2597	58	46	36%	179	30%	170	19%	25%	58	207	179	12	31	487	40%
i20c s25	10568	1023	58	44	37%	170	31%	159	20%	25%	52	185	158	8	21	424	41%
s11b cube cyl	5482	358	18	47	35%	190	33%	171	22%	28%	49	192	156	2	2	401	44%
i10c simp	8810	482	19	27	29%	116	27%	110	18%	22%	48	161	126	21	41	397	39%
i22u s27	60416	3456	94	36	27%	145	24%	144	16%	20%	42	158	148	6	16	370	31%
i15u s8	30430	1871	63	33	28%	130	24%	121	15%	20%	45	155	129	12	26	367	32%
i10u simp	18178	894	19	21	24%	97	23%	92	15%	19%	44	140	109	22	51	366	30%
i15c s8	6942	761	63	27	25%	110	22%	105	15%	18%	41	138	117	14	30	340	30%
i25c s40	7818	868	77	26	20%	112	18%	105	12%	15%	39	137	115	13	25	329	27%
s15b cylinder	3774	203	10	29	45%	121	43%	112	29%	36%	33	128	108	4	8	281	55%
n10u qtorus cyl	12146	403	6	8	11%	50	15%	50	10%	12%	30	83	58	22	48	241	19%
s08u cross cyls dr	5322	267	6	17	26%	80	27%	79	19%	22%	27	100	87	10	14	238	41%
i11u s1	34824	1528	27	20	42%	77	34%	75	22%	28%	25	89	81	5	11	211	45%
s08c cross cyls dr	3468	203	6	14	25%	63	24%	63	17%	20%	25	83	69	11	19	207	36%
i14c s7	5014	389	15	17	46%	66	38%	59	23%	30%	24	82	67	7	17	197	48%
i11c s1	7310	603	27	20	41%	81	35%	82	23%	29%	23	84	81	3	5	196	47%
n12b limit cycle genus0	5464	334	14	15	19%	82	24%	84	18%	21%	17	86	82	2	4	191	36%
s11c cube cyl	1926	174	10	16	42%	70	39%	69	26%	32%	19	72	66	2	1	160	51%
i14u s7	35038	1406	15	12	32%	51	29%	50	19%	24%	18	64	56	6	14	158	38%
s16c torus	4478	116	1	14	58%	61	54%	61	37%	45%	21	66	56	5	7	155	70%
s06b dodecahedron	1684	186	28	8	4%	53	6%	58	5%	5%	13	62	59	5	9	148	9%
n02b skijump anti box cyl	10288	588	22	4	3%	24	5%	22	3%	4%	18	43	27	14	30	132	6%
s05u cube sphere	11748	527	10	15	41%	57	35%	54	23%	29%	16	58	53	1	1	129	46%
n07c anti pyramid	4954	375	13	6	10%	37	13%	38	9%	11%	13	48	42	7	13	123	23%
n04b transition prism	2936	252	14	18	32%	58	24%	44	13%	19%	18	58	44	0	0	120	31%
s09b bridge	4200	323	22	13	16%	49	15%	42	10%	13%	15	52	42	2	1	112	22%
n09u pyramid	7844	342	5	10	40%	37	30%	29	16%	23%	16	46	32	6	8	108	37%
n10c qtorus cyl	5164	230	6	7	11%	35	13%	33	8%	10%	12	43	36	5	9	105	18%
n08u pentapyr	6042	352	6	6	21%	28	19%	27	13%	16%	13	40	30	7	15	105	23%
n09c pyramid	2324	160	5	8	32%	29	24%	23	13%	19%	13	39	28	5	10	95	30%
n03u skijump box cyl	4460	320	12	10	34%	39	29%	36	18%	24%	11	40	36	1	0	88	38%
n03c skijump box cyl	2188	196	12	9	32%	36	28%	34	18%	23%	10	37	34	1	0	82	36%
n06u anti pentapyr	1088	140	14	4	8%	22	9%	24	7%	8%	7	27	26	3	5	68	14%
n12u limit cycle genus0	7158	336	6	5	42%	22	39%	24	27%	32%	6	23	22	1	3	55	45%
n06c anti pentapyr	4914	367	14	2	3%	17	6%	22	5%	5%	4	20	23	2	4	53	10%
s05b cube sphere	36684	1128	18	8	9%	24	7%	18	4%	6%	8	24	18	0	0	50	10%
s17b sphere	3714	112	10	8	18%	24	14%	18	8%	11%	8	24	18	0	0	50	18%
s14b cube corner sub sphere	2648	216	18	7	11%	23	9%	16	5%	7%	8	24	16	1	0	49	12%
s15c cylinder	2362	110	2	5	50%	21	45%	22	31%	37%	5	20	20	0	0	45	55%
s15u cylinder	6896	223	2	5	50%	21	45%	22	31%	37%	5	20	20	0	0	45	55%
n12c limit cycle genus0	1780	139	6	4	33%	18	32%	20	23%	27%	4	17	18	0	0	39	39%
n04c transition prism	718	68	6	4	33%	16	28%	16	18%	23%	4	16	16	0	0	36	36%
n04u transition prism	7890	302	6	4	33%	17	30%	18	20%	25%	4	16	16	0	0	36	36%
s05c cube sphere	2748	205	10	4	15%	14	12%	12	8%	10%	4	14	12	0	0	30	16%
s17c sphere	1702	21	2	4	29%	14	25%	12	16%	21%	4	14	12	0	0	30	33%
s17u sphere	3420	32	2	4	29%	14	25%	12	16%	21%	4	14	12	0	0	30	33%
s07b notch	8902	523	20	3	5%	14	6%	12	4%	5%	3	14	12	0	0	29	8%
s06c dodecahedron	1626	152	20	1	2%	7	4%	7	2%	3%	3	10	8	2	3	26	4%
n02c skijump anti box cyl	2988	268	14	3	27%	10	19%	8	10%	14%	3	10	8	0	0	21	22%
n02u skijump anti box cyl	10686	571	14	3	27%	10	19%	8	10%	14%	3	10	8	0	0	21	22%
s09c bridge	4054	310	14	2	20%	8	16%	8	10%	13%	2	8	8	0	0	18	20%
s09u bridge	11652	618	14	2	20%	8	16%	8	10%	13%	2	8	8	0	0	18	20%
s11u cube cyl	9196	486	10	2	8%	7	7%	6	4%	5%	2	7	6	0	0	15	9%
n07u anti pyramid	21650	955	13	0	0%	3	1%	5	2%	1%	0	3	5	0	0	8	2%
n13b acute line	14122	561	14	0	0%	3	2%	4	2%	2%	0	3	4	0	0	7	3%
s10c cyl cutsphere	1140	94	5	0	0%	2	2%	4	4%	3%	0	2	4	0	0	6	5%
s10u cyl cutsphere	2706	182	5	0	0%	2	2%	4	4%	3%	0	2	4	0	0	6	5%
s07c notch	1442	149	12	0	0%	1	2%	2	3%	2%	0	1	2	0	0	3	4%
s07u notch	3750	276	12	0	0%	1	2%	2	3%	2%	0	1	2	0	0	3	4%

Table 4. Scaled Jacobian values (min and avg) of hexahedral meshes generated with a meshing pipeline using our collapsing and blockwise IGM remapping strategy versus a global IGM recomputation, LMFF-opt. Inputs are the non-trivial instances (those that actually require collapsing) from the third dataset. In line with [Liu and Bommes 2023], a resolution of 100,000 hexahedra was targeted in quantization. The same feature-preserving hex mesh optimization was applied in both cases, using the *Mesquite* library [Brewer et al. 2003] as described in [Liu and Bommes 2023]. Note that a valid IGM does not guarantee that the implied hex mesh always has geometrically valid elements when assuming (tri)linear elements.

Model	Ours avg	LMFF-opt avg	Ours min	LMFF-opt min
i01c m1	0.98	0.98	0.13	0.02
i01u m1	0.98	0.98	0.29	0.03
i02c m2	0.95	0.93	0.05	0.30
i02u m2	0.95	0.95	0.10	0.01
i06b m6	0.96	—	0.22	—
i06c m6	0.99	0.98	0.20	0.13
i06u m6	0.99	0.98	0.29	0.16
i09c m9	0.93	0.98	0.03	0.01
i09u m9	0.94	0.95	0.05	0.01
i10c simp	0.95	0.94	0.10	0.10
i10u simp	0.94	0.94	0.01	0.01
i11c s1	0.98	0.98	0.01	0.01
i11u s1	0.99	0.99	0.01	0.01
i12c s5	0.97	0.96	0.05	0.05
i12u s5	0.97	0.96	0.03	0.04
i13c s6	0.93	0.96	0.04	0.07
i13u s6	0.97	0.97	0.04	0.04
i14b s7	0.97	0.97	0.48	0.10
i14c s7	0.98	0.97	0.06	0.10
i14u s7	0.99	0.99	0.02	0.03
i15b s8	0.97	0.98	0.53	0.25
i15c s8	0.99	0.99	0.24	0.25
i15u s8	0.99	0.99	0.33	0.33
i17u s20	0.91	—	-0.13	—
i18b s22	0.97	0.97	0.43	0.14
i18c s22	0.98	0.98	0.09	0.11
i18u s22	0.99	0.98	0.06	0.07
i20c s25	0.97	0.97	0.01	0.00
i20u s25	0.98	0.98	0.02	0.02
i22c s27	0.98	0.98	0.03	0.04
i22u s27	0.99	0.98	0.05	0.04
i23c s31	0.96	0.97	-0.03	-0.11
i25c s40	0.97	0.97	0.05	0.11
i25u s40	0.97	0.97	0.04	0.03
n02b skijump anti box cyl	1.00	1.00	0.10	0.10
n02c skijump anti box cyl	0.99	0.99	0.10	0.10
n02c skijump anti box cyl	0.99	0.99	0.10	0.10
n02u skijump anti box cyl	1.00	0.99	0.04	0.03
n03b skijump box cyl	0.99	1.00	0.59	0.10
n03c skijump box cyl	0.99	0.99	0.08	0.10
n03u skijump box cyl	0.99	0.99	0.07	0.08
n04b transition prism	0.97	0.98	-0.92	-0.93
n04c transition prism	0.95	0.94	0.22	0.22
n04u transition prism	0.96	0.95	0.22	0.22
n05c box min pcyls	0.97	0.97	-0.66	0.14
n05u box min pcyls	0.98	0.98	-0.14	0.33
n06c anti pentapyr	0.87	0.96	-0.94	-0.71
n06u anti pentapyr	0.96	0.96	-0.42	-0.01
n07c anti pyramid	0.82	0.93	-0.77	-0.84
n07u anti pyramid	0.91	0.91	-0.82	-0.79
n08c pentapyr	0.91	0.89	0.04	0.24
n08u pentapyr	0.87	0.87	0.08	0.18
n10c qtorus cyl	0.96	0.93	0.19	0.00
n10u qtorus cyl	0.97	0.96	0.10	0.32
n12b limit cycle genus0	0.98	0.99	0.50	0.27
s04b tetrahedron	0.97	0.98	0.00	0.01
s05u cube sphere	0.99	0.99	0.37	0.38
s06b dodecahedron	0.98	0.98	0.28	0.28
s07u notch	1.00	0.99	0.68	0.72
s08b cross cyls dr	0.98	0.98	0.33	-0.93
s08c cross cyls dr	0.98	0.98	0.00	0.16
s08u cross cyls dr	0.99	0.99	0.00	0.26
s09b bridge	1.00	1.00	0.62	0.60
s09c bridge	1.00	1.00	0.87	0.92
s09u bridge	1.00	1.00	0.85	0.91
s11b cube cyl	0.99	0.99	0.51	0.49
s11c cube cyl	0.99	0.99	0.41	0.53
s15b cylinder	0.96	0.99	0.39	0.76
s15c cylinder	0.99	1.00	0.77	0.85
s15u cylinder	0.99	0.99	0.75	0.84
s16c torus	0.99	0.99	0.85	0.85
s17c sphere	0.99	0.99	0.68	0.71