Expansion Cones: A Progressive Volumetric Mapping Framework

VALENTIN Z. NIGOLIAN, University of Bern, Switzerland
MARCEL CAMPEN, Osnabrück University, Germany
DAVID BOMMES, University of Bern, Switzerland

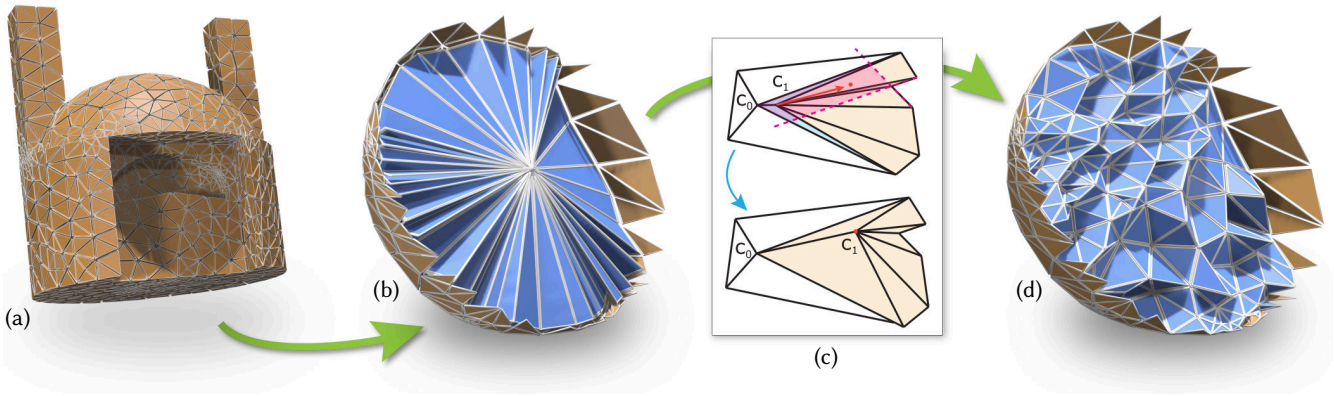

Fig. 1. Our method generates a bijective map of a ball-topology tetrahedral mesh (a) to a star-shaped domain, e.g. a ball (b). Starting with all interior vertices clustered inside the domain's kernel (b), we iteratively split clusters by picking a subset of vertices whose 1-ring neighborhood union has a non-empty kernel. By moving this subcluster into this kernel, some initially degenerate tetrahedra are expanded (c), without degenerating or inverting others. Repeating this until no cluster remains, all tetrahedron images obtain positive volume, yielding a bijective map (d). Mesh refinement is applied adaptively in the process to obtain the necessary degrees of freedom. A key invariant is that the intermediate maps never invert any tetrahedron.

Volumetric mapping is a ubiquitous and difficult problem in Geometry Processing and has been the subject of research in numerous and various directions. While several methods show encouraging results, the field still lacks a general approach with guarantees regarding map bijectivity. Through this work, we aim at opening the door to a new family of methods by providing a novel framework based on the concept of *progressive expansion*. Starting from an initial map of a tetrahedral mesh whose image may contain degeneracies but no inversions, we incrementally adjust vertex images to expand degenerate elements. By restricting movement to so-called *expansion cones*, it is done in such a way that the number of degenerate elements decreases in a strictly monotonic manner, without ever introducing any inversion. Adaptive local refinement of the mesh is performed to facilitate this process. We describe a prototype algorithm in the realm of this framework for the computation of maps from ball-topology tetrahedral meshes to convex or star-shaped domains. This algorithm is evaluated and compared to state-of-the-art methods, demonstrating its benefits in terms of bijectivity. We also discuss the associated cost in terms of sometimes significant mesh refinement to obtain the necessary degrees of freedom required for establishing a valid mapping. Our conclusions include that while this algorithm is only of limited immediate practical utility due to efficiency concerns, the general framework has the potential to inspire a range of novel methods improving on the efficiency aspect.

Authors' addresses: Valentin Z. Nigolian, valentin.nigolian@gmail.com, University of Bern, Bern, Switzerland; Marcel Campen, campen@uos.de, Osnabrück University, Osnabrück, Germany; David Bommes, david.bommes@unibe.ch, University of Bern, Bern, Switzerland.

0730-0301/2023/8-ART
https://doi.org/10.1145/3592421

CCS Concepts: • **Computing methodologies** → **Mesh models**; **Volumetric models**.

Additional Key Words and Phrases: tetrahedral mesh, Tutte embedding, bijection, homeomorphism, star-shaped

ACM Reference Format:
Valentin Z. Nigolian, Marcel Campen, and David Bommes. 2023. Expansion Cones: A Progressive Volumetric Mapping Framework. *ACM Trans. Graph.* 42, 4 (August 2023), 18 pages. https://doi.org/10.1145/3592421

1 INTRODUCTION

In discrete Geometry Processing, a fundamental problem is finding a continuous *map* between two objects. Most relevant are homeomorphic maps, which in addition are bijective, as is important, e.g., in the context of parametrization, mesh generation, or deformation. We consider here the problem of constructing such maps, from three-dimensional volumetric objects onto a certain class of three-dimensional domains. The object is assumed to be represented discretely by a tetrahedral mesh, and the map shall be piecewise linear (PL). In this setting, bijectivity implies that no mesh element is degenerated or inverted by the map.

While for the analogous 2D case, with triangle meshes in the plane or on surfaces, finding bijective maps is a well-documented problem with multiple solutions, it remains a challenge with only partial or limited solutions for 3D volumetric meshes. Among the various reasons for the increased intricacy of the problem in 3D is the higher relevance of the spatial discretization, the mesh structure. It is easy to generate examples of meshes that cannot be mapped bijectively into a desired shape (even if very simple, e.g. convex) with a PL map. This suggests that mesh structure modifications are important when the focus is on bijectivity. Hence, more precisely, we consider the

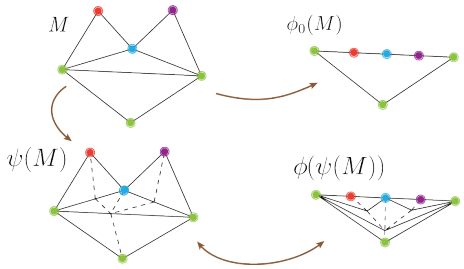

Fig. 2. 2D example of a mesh M and a given map ϕ_0 mapping boundary vertices (left) to desired positions (right), for which no bijective PL map for the interior exists. With an additional refinement map ψ, refining M through edges splits into M', we can find a bijective PL map ϕ satisfying the boundary conditions ϕ_0. While in this 2D illustration the necessity of refinement can only be demonstrated due to colinear boundary vertices and edges spanning the mesh, in 3D the issue is more common.

problem of finding a mesh modification $\psi(M) = M'$ that transforms the given tetrahedral mesh M into another tetrahedral mesh M' covering the same space, such that there exists a PL map $\phi : M' \to D \subset \mathbb{R}^3$, mapping M' onto the desired domain D (cf. Figure 2). Instead of *remeshing* M arbitrarily, we restrict to *refinement* to arrive at M'. This is to ensure there is a simple relation between M and M' (each element of M corresponds to and coincides with a union of one or more elements of M'), so as to minimize interfacing challenges with upstream and downstream processes.

Similar to many other works on mapping and parametrization, we consider a restricted class of domain shapes only. Concretely, we assume the desired volumetric domain D is *star-shaped*, as defined further below. Note that a bijection between two *arbitrary* shapes M_1 and M_2 (of ball-topology) can be defined as a composition of two maps $\phi_1 \to D$, $\phi_2 \to D$ via such a domain, namely as $\phi_2^{-1} \circ \phi_1$. In concurrent work [Hinderink and Campen 2023] the same problem setting as ours is addressed using an alternative approach.

1.1 Contributions

The main contribution of our work is a novel *progressive mapping* framework called Shrink-and-Expand (SaE) based on the concept of *expansion cones*. We see it as the root of a potential family of algorithms that produce maps from a ball-topology tetrahedral mesh to a star-shaped domain, exactly respecting a prescribed boundary map. It follows the idea of initializing the map such that all interior mesh elements are shrunk to a single common point. Then, the map can be incrementally modified so as to expand mesh elements, one-by-one or in a divide-and-conquer manner. Mesh refinement is performed on-the-fly, wherever necessary to maintain the invariant that no element is ever inverted by an expansion. Unless a specific topological condition is violated (cf. Section 4.3.1), the approach, by construction, yields a map that is a bijection.

Based on this framework, we furthermore describe a concrete prototype algorithm. It uses exact rational arithmetic; this allows ensuring correctness (though interfacing with floating point-based downstream applications is nontrivial cf. Section 6.2.1). We evaluate and benchmark it on a large dataset of ball-topology tetrahedral meshes, mapping to multiple types of domain shapes. We hasten to emphasize that this prototype is limited in terms of efficiency, in that for a portion of our benchmarking dataset it exceeds reasonable runtime. Nevertheless, we observe that it already clearly outperforms the state of art in terms of achieving bijectivity within a given a time limit.

The source code of our implementation and the dataset are available at https://www.algohex.eu/publications/expansion-cones/.

2 RELATED WORK

Mapping Triangle Meshes. For triangle meshes, guaranteed solutions for the construction of bijective maps to convex 2D domains can be found early in the literature [Tutte 1963; Floater 1997; Sheffer et al. 2006]. Extensions to broader domain classes [Kraevoy et al. 2003; Weber and Zorin 2014; Aigerman and Lipman 2015] or to maps between triangulated surfaces [Schreiner et al. 2004; Kraevoy and Sheffer 2004; Schmidt et al. 2019, 2020] followed, as did improvements in terms of numerical robustness [Shen et al. 2019].

Besides such constructive approaches, many forms of numerical optimization for map distortion minimization have been explored, e.g. [Hormann and Greiner 2000; Lipman 2012; Fu et al. 2015]. When initialized with a bijective (or injective) map, this property can be maintained in the process by means of barriers or constraints [Schüller et al. 2013; Bommes et al. 2013; Smith and Schaefer 2015; Rabinovich et al. 2017].

Mapping Tetrahedral Meshes. In contrast to the above wide range of solutions for the 2D case, the analogous 3D problem of finding a bijective volumetric map of a tetrahedral mesh onto a 3D domain remains a very difficult challenge. A discussion of some underlying reasons can be found in [Campen et al. 2016]. In particular, several fundamental theorems that the above 2D methods are directly or indirectly based on do not extend to 3D, or at least not in a simple manner. A notable example is a discrete version [Floater 2003] of the Radó-Kneser-Choquet theorem, stating that a convex combination map between any disc-topology triangle mesh and the unit disc is a bijection. This result for 2D meshes unfortunately does not hold in 3D [Floater and Pham-Trong 2006; Alexa 2023].

Reviewing the literature, an everlasting conflict can be observed between robustness (achieving injectivity or bijectivity), efficiency (runtime), and constraint satisfaction (specification of the target shape) of 3D mapping methods. Methods commonly perform well in one or two but not all three of these aspects. For instance, [Liao et al. 2021], which deals with maps for the purpose of deformation, guarantees local injectivity, but not the satisfaction of positional constraints. By contrast, [Xia et al. 2010] enforces constraint satisfaction, but guarantees bijectivity only for a very restricted class of input shapes. Likewise, [Aigerman and Lipman 2013] can enforce positional constraints and is fast, but cannot guarantee yielding an injective map. Finally, [Campen et al. 2016] provides guarantees regarding both constraint satisfaction and bijectivity, but often requires heavy mesh refinement to output a piecewise linear map representation, impeding heavily on run time performance. This latter work highlights, though, that mesh refinement can be an important ingredient to achieve robustness with respect to discretization.

Recently, there has been a certain focus on numerical optimization-based methods, aiming at injectivity by minimizing the amount of map inversions, often in combination with lowering map distortion [Fu et al. 2015; Su et al. 2019; Du et al. 2020; Garanzha et al. 2021]. However, finding a formulation with a smooth and convex objective, such that the global optimum can be found, and at the same time guaranteeing that this optimum is injective remains a difficult challenge. Even worse, assuming a fixed discretization, as given by the input mesh, such a map may not even exist, unless some form of refinement is employed. While mesh refinement has been considered in various works on the 2D problem [Kraevoy et al. 2003; Lee et al. 2008; Jin et al. 2014; Shen et al. 2019; Gillespie et al. 2021; Campen et al. 2021], to the best of our knowledge the only 3D mapping method systematically using any such operations for the purpose of ensuring feasibility is [Campen et al. 2016], albeit not in an numerical optimization-based context. Some further methods make use of more general remeshing in the context of 3D deformation-oriented mappings [Wicke et al. 2010], e.g. to improve simulation quality [Narain et al. 2012; Anderson et al. 2005].

Progressive Embedding. Our work was in part inspired by the progressive embedding idea [Shen et al. 2019], which allows yielding bijective 2D maps onto planar domains. First, given an initial map, it collapses edges until all inverted elements are gone. Then, in essence, those collapses are undone in reverse order, positioning the reintroduced vertices in the domain in such a way that no degenerate or flipped triangles emerge, ultimately yielding the desired map. In an extreme case, it may be necessary to collapse until only a single interior vertex remains. Such collapse sequences are known to always exist for disc-topology triangle meshes under mild conditions. Likewise, the ability to find a valid position for each reintroduced vertex is shown to be guaranteed.

However, neither of these two properties hold in an analogous 3D setting [Livesu 2020]. Indeed, edges that do not satisfy the link condition are not collapsible [Dey et al. 1999], and while this can easily be circumvented in 2D [Shen et al. 2019], the range of possible blocking configurations drastically increases in 3D. Although for any tetrahedral mesh there exists a finite sequence of barycentric subdivisions that guarantees to make its full collapsing possible [Adiprasito and Benedetti 2020], there is no efficient known way to find such a sequence. Regarding the second stage, one can easily find examples of a vertex that cannot be reintroduced at any position without creating degenerate or inverted tetrahedra, as shown in Figure 3. On a conceptual level, our approach can be viewed as not topologically collapsing but geometrically shrinking all interior vertices to a common point (thereby circumventing the above obstacle), and then expanding vertices again, albeit not in a fixed order and assisted by systematic mesh refinement and map warping (thereby yielding valid positioning options).

3 OVERVIEW

First, we introduce a set of underlying concepts for our method. Since all concepts these are well-suited for both 2D and 3D, we will often give 2D visual examples as they are sufficient to express most concepts and easier to read.

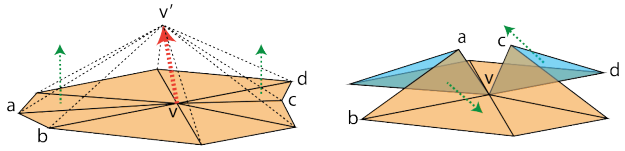

Fig. 3. Left: a vertex v' (formerly collapsed onto v) is reintroduced and positioned according to the red arrow; the thus reintroduced tetrahedra around the edge (v, v') get a positive volume. Right: here, by contrast, the normals of the faces (v, a, b) and (v, c, d) (green arrows) are opposite. Regardless of where v' would be positioned, either the tetrahedra (v, a, b, v') and (v, c, d, v') would both be degenerate or one of them would be inverted. The bottom sides of the triangles are colored blue for clarity.

3.1 Fundamentals

Star-Shapedness: A domain D is *weakly star-shaped* if and only if there exists a *guard*, a point $s_0 \in D$ such that for every point $s \in D$ the straight line segment $\overline{s_0 s}$ lies entirely inside of D. The set of all guards is called the *kernel* of D. A guard in the interior of the kernel (i.e. not on its boundary) will be called a *center* of D. The domain is *strongly star-shaped* if the kernel in addition is non-degenerate, i.e. it has a non-zero volume [Hansen et al. 2020] and thus has a center. Herein we make use of the strong notion only and will omit the qualifier for brevity. Note that convex domains are a subset of star-shaped domains, with the kernel being the entire domain.

Refinement Map: We will perform mesh refinement by means of the *edge split* operator. By splitting an edge $e = (a, b)$, its is replaced by two edges (a, c) and (c, b), and all tetrahedra (a, b, i, j) are split into two tetrahedra (a, c, i, j) and (c, b, i, j). By default, this new vertex c, which we call a *mid-vertex*, is initially located at the midpoint between vertices a and b. We call a sequence of such splits a *refinement map*, denoted by ψ in the following.

3.2 Approach

Input: Our method takes a tetrahedral mesh M of ball-topology, embedded in $\mathbb{R}^3$ such that each tetrahedron (also called *cell*) has a positive volume. Its boundary is denoted ∂M. The set of vertices is $V = V^\circ \cup V^\bullet$, with boundary vertices V° and interior vertices $V^\bullet$. We use M to denote the mesh as well as the subset of $\mathbb{R}^3$ it occupies; the distinction will be clear from the context. In addition, an injective orientation-preserving boundary map $\phi_0 : \partial M \to \mathbb{R}^3$ is taken as input, such that $\phi_0(\partial M)$ is the boundary of a star-shaped domain D. ϕ_0 is specified in terms of boundary vertex images, $\phi_0(V^\circ)$, and interpreted as linear over the boundary triangles.

Output: Our goal is to find a map $\phi : M' \to D \subset \mathbb{R}^3$, linear per tetrahedron of a possibly refined version M' of M, that conforms with the boundary map ϕ_0 and is injective. This implies it is bijective onto the star-shaped domain D defined by ϕ_0. M' is obtained from M by means of a refinement map ψ, a sequence of edge splits only. Their relation thus is simple, they are nested, and the map ϕ is also piecewise linear on M, but with multiple linear pieces per tetrahedron. Both ψ and ϕ are crafted with a sequence of steps carefully designed to ensure the bijectivity of ϕ.

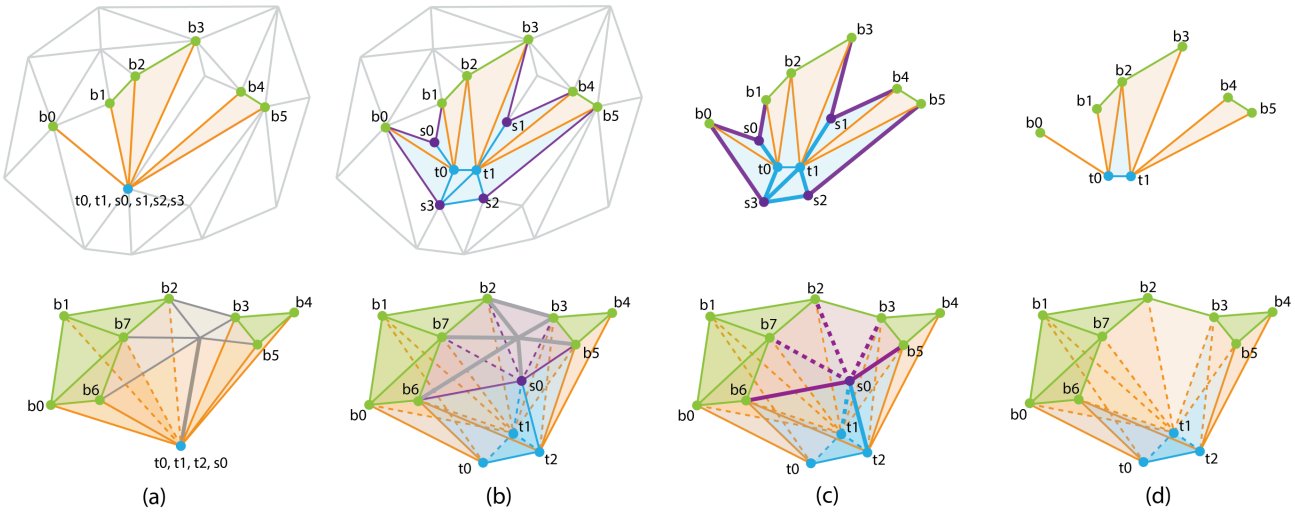

Fig. 4. 2D (top) and 3D (bottom) example of a cluster $\{s_i, t_i\}$ and the expansion cone of its subcluster $\{t_i\}$. Tip vertices are labelled t, base vertices b and other submesh vertices s. From left to right we have (a): the expansion cone of the subcluster $\{t_i\}$ inside a small mesh (most of the 3D mesh is omitted to avoid clutter). (b): coincident vertices are visually pulled apart to show the degenerate simplices. (c): just the submesh. (d): just the expansion cone. Simplices are color-coded as follows: blue means degenerate, i.e. incident to at least two coincident vertices, orange means "spoke", i.e. connecting the tips and the base, green means base, purple means submesh boundary and gray is the rest. Notice that in both examples the expansion cone is not of ball-topology (disc-topology in the 2D case).

Initialization: As a first step of our method, we pick an interior guard c of D, set $\phi(v_i) = c, \forall v_i \in V^\bullet$ and $\phi(v_i) = \phi_0(v_i), \forall v_i \in V^\circ$. This initial map conforms with ϕ_0 on the boundary, and maps all interior vertices to a common point in the interior. This initial map is similar in spirit to that of Shen et al. [2019] in the 2D case, with the major difference that we do not topologically (in terms of mesh connectivity) but only geometrically collapse the inner elements, circumventing the topological issues discussed in Section 2.

Invariants: An important property of the initial map is that it, while degenerating many tetrahedra, does not invert any tetrahedron. This will be an invariant in the following construction. Furthermore, all degenerate tetrahedra are degenerate due to coincident vertices, not to noncoincident colinear or noncoincident coplanar vertices. This likewise will be an invariant. Starting from this initial situation, our goal is to step-by-step find degenerate tetrahedra whose volume can be made positive by pulling coinciding vertices (more precisely: their images under ϕ) apart within D.

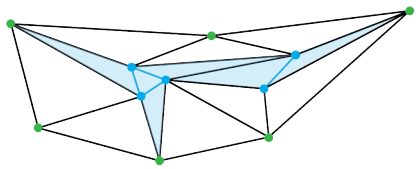

Fig. 5. A 2D mesh showing unexpanded (clustered) vertices (blue) and expanded vertices (green). The zero-length edges (connecting two coincident vertices) and zero-area triangles (with at least two coincident vertices) are drawn slightly expanded (in blue) for visualization purposes.

3.3 Clusters & Cones

Cluster: We call *cluster* a set of interior vertices which are coincident under (a current state of) map ϕ. A cluster containing k vertices is called a k-cluster (with $k \geq 1$). Assuming an arbitrary ordering, we denote as C_i the i-th cluster. The cluster containing a given vertex $v \in V^\bullet$ is denoted $C(v)$. A vertex is called *expanded* if it forms a 1-cluster, $C(v) = \{v\}$, otherwise *unexpanded*. Initially, there is only one cluster $C_0 = V^\bullet$. See Figure 5 for a 2D illustration of this concept. A *subcluster* is a strict subset $S \subset C$ of a cluster C. Its *complement* is denoted $\bar{S} = C \setminus S$. 1-(sub)clusters are called *trivial*, while k-(sub)clusters, for $k > 1$, are called *non-trivial*.

Cluster Split: On the highest level, our framework makes use of a single operation that is applied repeatedly: the *cluster split*, splitting a cluster into two. The algorithm terminates once all clusters are trivial, i.e. when all vertices are expanded. A cluster C is split by selecting a subcluster $S \subset C$ and translating all vertices of S by a common vector t, maintaining their coincidence: $\phi(S) \leftarrow \phi(S) + t$. Afterwards the former subclusters S and $\bar{S}$ form two smaller separate clusters. The challenge lies in finding a subcluster S and a vector t such that this maintains our invariants, most importantly that no tetrahedron is inverted. To find these, we make use of the following concepts.

Subcluster Mesh: The *subcluster mesh* (or *submesh* for short) of a subcluster S, denoted as $Submesh(S)$, is the closure of the set $\{\sigma \in M | \sigma \cap S \neq \emptyset\}$, the set of all simplices that are incident to at least one vertex of S. Closure here means that also their subsimplicies are included; i.e. a tetrahedron's triangles, a triangle's edges, an edge's vertices. Informally, it is the union of the subcluster vertices' direct neighborhoods. Figure 4c shows an example of a subcluster mesh.

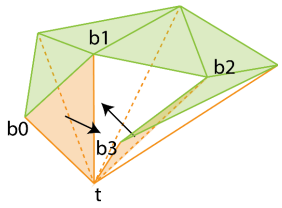

Fig. 6. 3D example of a topologically-expandable subcluster $\{t\}$ (its expansion cone has ball-topology and a disc-topology base) that is not simply-expandable. The two orange spoke faces (t, b_1, b_0) and (t, b_3, b_2) have normals such that the expansion cone is not star-shaped. Any movement of the tip vertex t would invert at least one of the degenerate tetrahedra (not shown) that surround the cone's outer spokes by definition.

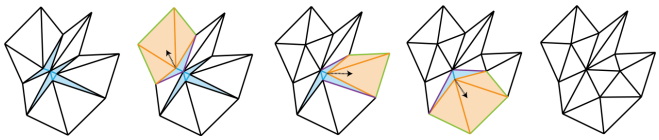

Fig. 7. 2D example of a simple progressive expansion process. We start from the leftmost mesh with all 4 interior vertices at the same location (the blue simplices are actually degenerate) and then expand the clustered vertices one by one. The three middle images show the intermediate steps, with the iteratively expanded vertex' expansion cone highlighted. The cone base is in green, spokes in orange. The arrows show the translation vectors t of those vertices within their expansion cones' kernels. The rightmost image shows the result, with all triangles having positive area. The last vertex does not need to be expanded since the cluster has already become trivial.

Expansion Cone: The *expansion cone* of a subcluster S of cluster C, denoted $EC(S)$, is the subcluster mesh $Submesh(S)$ with all simplices removed that are incident to a vertex of the complement $\bar{S}$. Concretely: $EC(S) = Submesh(S) \setminus \{\sigma \in M | \sigma \cap (\bar{S}) \neq \emptyset\}$. Figures 1c and 4d show examples of expansion cones. We call the vertices of S the *tips* of $EC(S)$ and all simplices of $EC(S)$ that are not incident to any vertex of S the *base* of $EC(S)$. Simplices incident to the base and the tips are called *spokes*. The reason for defining the expansion cone is that it tells us how the translation vector t can be chosen such that the cluster split of S from C along t maintains the above invariants. Namely, to that end, we need to choose t such that $\phi(S) + t$ lies *inside the kernel of* $EC(S)$, as illustrated in 2D in Figure 1c. In this sense, the expansion cone defines how the subcluster can validly be expanded from its cluster.

Expandability: A subcluster whose expansion cone is star-shaped is called *simply-expandable*, because it immediately allows for a cluster split. Any center of the cone (a point in its kernel's interior) can be chosen as target. If a cone is not star-shaped but has ball-topology (and a disc-topology base), the obstacle to expandability is of geometrical kind only; we call the corresponding subcluster *topologically-expandable*, see Figure 6. If it does not even have this topology (as in the examples in Figure 4d), the subcluster is *unexpandable*. In Appendix A the topology check is detailed. Topologically-expandable subclusters can be made simply-expandable through careful deformations of the neighborhood, i.e. adjustements to $\phi(Submesh(S))$, possibly assisted by mesh refinement.

4 PROGRESSIVE EXPANSION FRAMEWORK

Before we give a complete and generic description of our framework, let us get a better grasp of the idea, based on the 2D example shown in Figure 7. For simplicity, let us assume for a moment that we can always find a *trivial* subcluster (of the initial cluster C_0) that is *simply-expandable*. For this subcluster, we then perform a cluster split, expanding the subcluster's single vertex to a center of its expansion cone. Note that this strictly reduces the global coincidence number, the number of vertex pairs (v, w), $v \neq w$, with $\phi(v) = \phi(w)$. This is repeated until all vertices have been expanded and C_0 becomes trivial, with only one vertex left. This means the coincidence number is zero, which, due to our invariants, implies that all tetrahedra have positive volume under ϕ.

If no trivial subcluster is simply-expandable at some point, we can also choose a simply-expandable larger (non-trivial) subcluster for expansion. This likewise has the desired strictly monotonic effect on the coincidence number. This results in two non-trivial clusters, which are then both treated analogously to the single non-trivial cluster C_0 in the above example.

4.1 Star-Shapification

Unfortunately, in the general setting, simply-expandable subclusters (whether trivial or non-trivial) do not always exist. In such a case we fall back to a topologically-expandable subcluster. While this does not allow for expansion right away, we can warp the mesh's image under the map around the subcluster so as to make the expansion cone star-shaped and thus the subcluster simply-expandable. In such a case, we say the expansion cone is *star-shapified*. While this is obviously possible with a smooth continuous deformation, we wish to remain in a discrete PL setting. Through mesh refinement the degrees of freedom can be obtained that are necessary to express such a warp piecewise linearly while maintaining bijectivity. With such a procedure available, we can effectively consider all topologically-expandable cones as simply-expandable, up to *star-shapification*. In Section 5 we describe one concrete option to implement this operation.

4.2 Isolation

After splitting a cluster into two non-trivial clusters, these two are directly adjacent. This implies that for at least some of their subclusters, the expansion cone is incident to the other cluster. Hence, their bases contain degenerate elements—which is not the case for the initial cluster or when expanding trivial subclusters. In order to avoid the need for special handling of such more complex expansion cone structures, we opt to apply one additional operation after splitting a cluster into two non-trivial clusters. Namely, we *isolate* the two clusters from each other in terms of direct connectivity, essentially by splitting the edges that connect them. Afterwards, each cluster has the same structural characteristics like the initial cluster, in that there are no coincident vertices in its 1-ring. In Section 5 we describe one concrete option to implement this operation.

Algorithm 1 Our generic Shrink-and-Expand framework

Input:
M, a ball-topology tetrahedral mesh
ϕ_0, a homeomorphic boundary PL map onto a star-shaped domain D
Output:
ψ, a refinement map of M, yielding M'
ϕ, a homeomorphic volumetric PL map of M' onto D

1: **procedure** ProgressiveExpansion(M, ϕ_0)
2: $\quad\phi(V^\circ) \leftarrow \phi_0(V^\circ)$
3: $\quad\phi(V^\bullet) \leftarrow$ Center(D)
4: $\quad$**while** $|C(v)| > 1$ for any v **do**
5: $\quad\quad S \leftarrow$ any topologically-expandable subcluster ▷ if none → fail
6: $\quad\quad(\phi, \psi) \leftarrow$ Star-Shapify(S)
7: $\quad\quad\phi(S) \leftarrow$ Center($EC(S)$)
8: $\quad\quad(\phi, \psi) \leftarrow$ Isolate($S, \bar{S}$)
9: $\quad$**end while**
10: **end procedure**

4.3 Summary

Algorithm 1 summarizes our generic framework. The subroutines

- Center (selection of a center from a kernel),
- Star-Shapify (map modification to make a cone star-shaped),
- Isolate (separate adjacent non-trivial clusters), as well as
- the choice of the next subcluster in each iteration

can be implemented in many different ways. In the following we describe and evaluate one concrete way.

4.3.1 Caveat. The algorithm succeeds in producing a bijection if the mesh M satisfies the topological condition that topologically-expandable subclusters are always available. If this is not the case, the algorithm will necessarily fail (line 5). In our experiments we have not been able to identify examples of clusters in which every single subcluster was unexpandable, but do not have a proof of general feasibility either.

5 IMPLEMENTATION

This section describes a concrete realization of our framework by discussing specific solutions for the aforementioned subproblems.

5.1 Subcluster Search

For reasons of efficiency, it makes sense to explore subclusters in order of increasing size, starting with the set of all 1-subclusters $\mathcal{S}_1$ of all clusters. Furthermore, because subclusters whose submesh is not connected are unexpandable, we form larger subclusters incrementally out of smaller ones, adding vertices from their 1-rings: $\mathcal{S}_{k+1} = \{S \cup \{v\} \mid S \in \mathcal{S}_k, n \in N_1(S)\}$, where $N_1(S)$ is the set of vertices from $\bar{S}$ (the remainder of the cluster) that are directly adjacent to a vertex in S.

Considering that an element of $\mathcal{S}_{k+1}$ can be built out of multiple $S \in \mathcal{S}_k$, algorithmically we employ a hash map to filter duplicates, avoiding testing the same subcluster for expandability multiple times. The ordered list of a subcluster's vertex indices is used as key.

While this simple strategy suffices, there might be more efficient ways to enumerate the connected subclusters, different orderings that lead to finding an expandable subcluster more quickly, or even ways to find expandable subclusters without exhaustive enumeration and testing.

Experimentation led to the conclusion that, for overall efficiency, it is advisable to further refine the order in which subclusters are considered for expansion. Concretely, we give priority to subclusters as follows: simply-expandable k-subclusters for $k = 1, 2, 3$ (in this order), followed by topologically-expandable 1-subclusters with small expansion cones (≤ 30 tetrahedra), simply-expandable k-subclusters for $k = 4, 5$, and finally topologically-expandable k-subclusters, in the order of increasing k.

5.2 Center Choice

For the choice of a center, i.e. an interior guard, from a star-shaped set (whether the domain D or an expansion cone $EC(S)$), we opt to compute the Chebyshev center, the interior point with maximum distance to the set's boundary. This is easily done by means of a linear program, maximizing the distance to all the boundary triangles' supporting plane.

While any center could be chosen without affecting the algorithm's correctness, the rationale for this specific choice is that we expect this maximally centered choice to typically be well-behaved in terms of numerics. Expanding a subcluster to this position expands the incident degenerate tetrahedra, which become non-degenerate, the most. This also leaves more room for adjacent clusters that still need to be expanded in subsequent steps.

5.3 Star-Shapification

Assuming only a topologically-expandable subcluster is available, we need a way to make it simply-expandable. We will first consider trivial subclusters (with a single tip vertex), and turn to non-trivial subclusters in Section 5.3.1.

Let us focus for a moment on the expansion cone in isolation, ignoring the mesh around it. We can deform it into star-shape without degenerating or inverting any contained elements as follows. We pick an arbitrary (e.g. centermost) vertex from the base of the expansion cone, the designated *witness vertex* w, and then "collapse" the whole expansion cone to the 1-ring neighborhood of the edge (w, t), with t being the tip of the expansion cone. This can be done incrementally, triangle by triangle of the cone's base in an outside-in manner, moving a vertex inwards so as to (temporarily) degenerate the triangle's spoke tetrahedron. This process is illustrated in Figure 8. The collapsed expansion cone necessarily is star-shaped, as the edge (w, t) lies in its kernel. However, it now contains multiple degenerate tetrahedra (those that were collapsed onto the 1-ring).

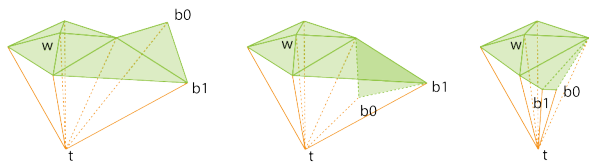

Fig. 8. By iteratively "collapsing" base vertices b_0 and then b_1 to $\partial N_1((w, t))$, with w being the "witness" vertex, we deform $EC(t)$ to become star-shaped. Note that this degenerates two previously nondegenerate tetrahedra of $EC(t)$. A subsequent "contraction" will expand them again.

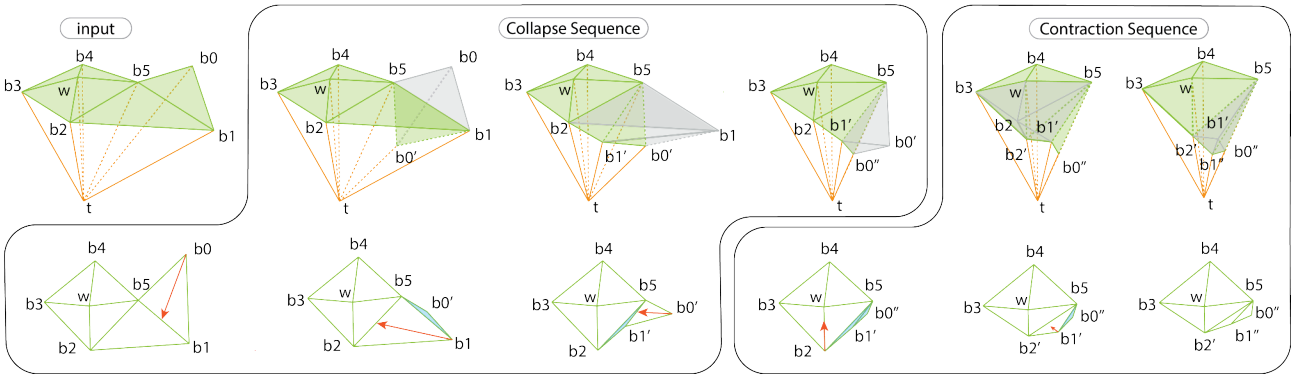

Fig. 9. Top: A 3D expansion cone and its evolution through a short star-shapification process. Bottom: A 2D projection view of this, showing the base, with the movement of vertices indicated by red arrows; the blue triangles are shown for visualization but are actually projected degenerate tetrahedra. For a given SS $(b_0 \rightarrow b_1)$, obtained from the initial expansion cone (left), we first perform, from left to right, the Collapse Sequence as follows: The edge (t, b_0) is split, creating vertex b_0', which is "collapsed" onto spoke face (t, b_5, b_1). Then, edge (t, b_1) is split, creating b_1', which is collapsed onto spoke face (t, b_5, b_2). Since b_0' was collapsed onto a face incident to b_1, its spoke edge t, b_0' is also split, creating vertex b_0''. Finally, b_0'' is collapsed onto face (t, b_5, b_1'). The grey parts are kept for visualization but are not part of the expansion cone, cf. Figure 10. The result is that vertices b_0'' and b_1' are collapsed to the boundary of $N_1((w, t))$ and $EC(t)$ is star-shaped. The last two steps are the Contraction Sequence: First, edge (b_2, t) is split, creating b_2', which is then moved towards the (w, t) axis, making its incident cells non-degenerate. This contraction step is repeated for the spoke (b_1', t). All cells of $EC(t)$ are now non-degenerate.

We therefore then slightly "contract" the base vertices, radially towards the edge (w, t) in inside-out order, so as to incrementally expand the temporarily degenerated tetrahedra, while maintaining star-shapedness. This yields a simply-expandable situation.

Of course, however, we must also take the mesh surrounding the expansion cone into account. The above procedure, moving base vertices around, might easily invert adjacent tetrahedra. There may just not be enough room for the required amount of movement. Hence, if a base vertex b cannot be moved as desired, we perform a split of its spoke edge (t, b), adding mid-vertex b'. Note that this reduces the extent of the expansion cone of tip t, as now b' is a base vertex, while b is no direct neighbor anymore. At the same time, this new base vertex b' now has freedom to move, within its 1-ring neighborhood, without causing inversions. This is illustrated in Figure 10.

Figure 9 gives a complete example of the overall process on a simple mesh. Implementation details on each step can be found in Appendices B.1 (determining a collapse order), B.2 (collapsing elements, assisted by spoke splits), and B.3 (contracting the star-shaped expansion cone to restore positive volume).

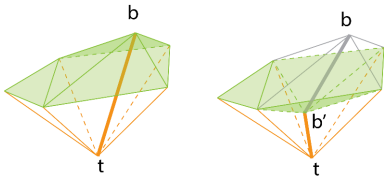

Fig. 10. By splitting a spoke edge going from the tip vertex t to a base vertex b and adding a new vertex b' between them, we effectively remove vertex b and its incident tetrahedra from the expansion cone, as b is no longer part of any simplex incident to t. This new base vertex b' is now free to move without creating inverted elements, as long as it moves within the kernel of its 1-ring neighborhood.

5.3.1 Nontrivial Subcluster Star-Shapification. So far we considered only trivial subclusters, i.e. expansion cones with a single tip. In the case of multiple tips, the above star-shapification approach does not apply as-is. This is because we assumed that each base vertex is connected to the tip with a unique spoke edge. In the case of multiple clustered tips, by contrast, a base vertex may be connected to the tips via multiple edges, of degenerate spoke *faces* or *tetrahedra*. Instead, we will first "simulate" this process by star-shapifying an expansion cone consisting of the same base but with a single tip. All spoke splits will then be duplicated across the degenerate faces and tetrahedra represented by single edges in the single-tipped copy of the subcluster expansion cone. The details are left to a formal description of this duplication process, given by Algorithm 6. Figure 11 shows a small example of this additional care.

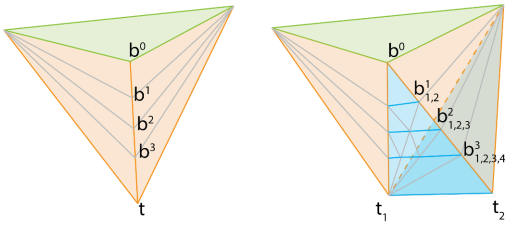

Fig. 11. A series of spoke splits on a 1-subcluster expansion cone (left) and their equivalent on a 2-subcluster expansion cone (right). Each split $(b^k, t) \rightarrow b^{k+1}$ is propagated on any previously-created edge (b_j^k, t_i), where $\{t_i\}$ is the set of cluster vertices. More splits are required after every spoke split to ensure that the "layers" of star-shapification remain isolated, i.e. to ensure that there is no edge (b^k, b^{k+m}) with $m > 1$ or (b^k, t_i) where $k < n$ and the edge (b, t) was split n times. Note that the interior edges have been omitted for readability reasons and that as usual, the light blue simplices are degenerate. Each set of vertices $\{b_j^k\}$ creates a new cluster that subsequently needs to be expanded.

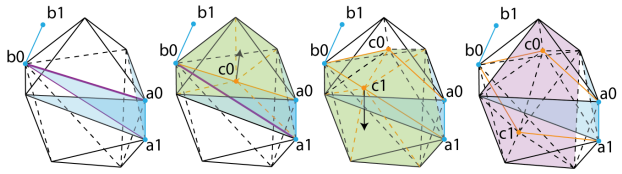

Fig. 12. A small example of cluster isolation between two 2-clusters $\{b_0, b_1\}$ and $\{a_0, a_1\}$ (left). As usual, degenerate simplices are colored blue. The first step is to split $(a_0, b_0) \rightarrow c_0$ and then expand the new mid-vertex within its expansion cone (center left). We then repeat this process for (a_1, b_0) (center right) and thus isolate the two clusters from each other. The rightmost figure highlights the faces making up the newly-created buffer surface between the two clusters (purple).

Note that the duplication of a split across degenerate spoke faces and tetrahedra creates a new cluster. Importantly, this type of cluster is simply-expandable (Appendix C.1). Hence, its subsequent expansion will not require star-shapification, thus will not introduce new clusters, thus will not lead to potentially infinite recursion. Instead of using the standard generic expansion routine (including isolation), a simplified routine (see Algorithm 6) specialized to this type of clusters proved to be sufficient and efficient in our experiments, though a fall-back to the standard routine may be necessary in general.

5.4 Cluster Isolation

After splitting a cluster C into two clusters C_a and C_b which both are nontrivial, there may be multiple edges connecting these. In particular, a subcluster $S \subset C_a$ may have an expansion cone whose base contains multiple vertices from C_b. The base then contains degenerate elements—which is in contrast to the requirements of our star-shapification approach from Section 5.3. We could generalize the approach to handle such configurations, but opt for the simpler strategy of resolving them beforehand.

This is done by isolating the two clusters by means of a buffer layer of vertices. To this end we simply split all edges connecting the two clusters. This results in a cluster of vertices that is simply-expandable (Appendix C.1), analogous to the clusters created by splitting multiple coincident spoke edges in Section 5.3.1. After their expansion, C_a and C_b are no longer directly adjacent. This process is illustrated by Figure 12 and implementation details are given in Appendix C.

5.5 Numerics & Efficiency

To avoid issues due to limited numerical precision, we make use of an infinite precision representation, for which we use CGAL's rational numbers [Fabri and Pion 2009]. This allows safely finding centers, exactly testing star-shapedness, and properly dealing with exact coplanarity or colinearity that occur temporarily in the star-shapification procedure (Appendix B.2).

Note that the exact representation of the Chebyshev center or the mid-vertices resulting from splits is increasingly expensive (in terms of memory and runtime) the more nested operations occur. Minimizing the overall required precision and the number of splits are thus important aspects of our implementation.

5.5.1 Adaptive Precision Reduction. The first way we can limit the growth of the number representation is to systematically reduce the precision of newly computed vertex positions, without however invalidating the required properties (no inversions, etc). Indeed, we can truncate the precision of a computed center, as long as it remains a center, i.e. inside the respective kernel. Our approach here is to perform a binary search, starting from the precision of the exact representation, to find the lowest precision that is sufficient. We perform a maximum of 10 iterations, and stop when the interval drops below 32 bytes. Those specific values were obtained empirically and represent a good trade-off between the effort required to reduce the necessary precision and the long-term benefit for the total expansion performance. In our experiments, this mechanism was sufficient to make a significant difference in terms of run time, with factors beyond 10×.

5.5.2 Vertex Relaxation. Another very helpful method is to periodically run a relaxation pass, moving some expanded, non-boundary vertices to the barycenter of their 1-ring neighbors as long as no inversions are created. The precision of the newly computed barycenter is also systematically reduced, the same way as described in the previous section. The choice of vertices subjected to this relaxation can have a drastic influence on performance. One could for instance limit the relaxation to the k-ring neighborhood of the remaining unexpanded vertices, or, simply smooth all interior expanded vertices. The goal is then to find a trade-off between the amount of time spent on relaxation versus the time gained running the next steps of the progressive expansion algorithm. Indeed, some expansion cones that are initially only topologically-expandable tend to become simply-expandable after relaxation. From our experiments, relaxing the 1-ring neighborhood of the remaining unexpanded vertices already increases the number of simply-expandable expansion cones. Therefore, we periodically run such a relaxation pass during our algorithm to improve its run time.

5.5.3 Edge Collapsing. Some operations, such as star-shapifying large expansion cones, often require many splits and thus create a large number of additional vertices. Those can then be part of the expansion cones of other remaining unexpanded vertices, making it increasingly costly to handle these. We thus add an additional clean-up pass, removing all added vertices by means of halfedge collapses that can be removed without creating new degeneracies or inversions. For a vertex c created by a split of edge (a, b), only halfedges (c, a) or (c, b) are considered, so as to maintain that M and M' are nested. Again, orders of magnitude of speed-up were observed for complex instances, especially for those requiring many star-shapifications.

5.6 The Prioritizing Expansion Algorithm

With all components described, we can now assemble a concrete algorithm based on our framework, in particular making use of the subcluster prioritization laid out in Section 5.1. Besides instantiating the subroutines of Algorithm 1 with the concrete algorithms described in Sections 5.1, 5.2, 5.3, and 5.4, we add the following details discussed in Section 5.5:

- Whenever new vertex image coordinates are computed (centers, relaxed positions), adaptive precision reduction (Section 5.5.1) is performed.
- Whenever an expansion moved a nontrivial subcluster or involved star-shapification, a relaxation pass (Section 5.5.2) and a cleanup pass (Section 5.5.3) is performed, restricted to the 1-ring neighbors of unexpanded vertices.

In the following we evaluate this algorithm, in particular its runtime and refinement behavior, and compare against alternative volumetric mapping methods.

6 EVALUATION

In the following we evaluate and analyze the behavior of our Prioritizing Expansion version of the Shrink-and-Expand (SaE) framework. We furthermore compare it to related state-of-the-art methods, two based on numerical optimization, TLC [Du et al. 2020] and FoF [Garanzha et al. 2021], and one based on a combinatorial construction, FOL [Campen et al. 2016].

6.1 Dataset

While TLC comes with a benchmark dataset, it focuses on the 2D case; the 3D subset is quite small and strongly biased towards one particular shape. Aiming at benchmarking our method on more diverse and challenging problems, we make use of the *TetWild* dataset of tetrahedral meshes [Hu et al. 2018]. The subset of ball-topology meshes consists or 3007 meshes, ranging from 156 to 1,283,106 tetrahedra. As a pre-processing step, so as to enable our initial map, we split interior edges connecting two boundary vertices, as well as interior faces whose three edges are boundary edges. For each of those meshes we then construct four types of boundary maps ϕ_0 as follows, so as to obtain input instances for the methods:

- Tetrahedron (T): We pick four boundary vertices randomly and find the shortest path between each pair of vertices. Those vertices are then mapped to the four corners of a unit tetrahedron centered at the origin and the paths to its edges, with uniform distribution. The remaining boundary vertices are mapped to the tetrahedron's four faces with a Tutte embedding. Note that depending on the random choice the result may be invalid (coincident paths, implied degeneracies). In such cases further attempts are made; if after 100 attempts no valid boundary map is obtained, the mesh is dropped from the dataset. This was the case for 29 meshes in total.
- Stiff Tetrahedron (ST): The idea is the same as for the previous one, except we start by picking one face of the initial mesh and mapping it to a face of the target tetrahedron. The fourth vertex is still picked at random. This results in a mesh that is geometrically more constrained closer to this initially-mapped face, hence the inherent "stiffness" of this boundary. Here 30 meshes are dropped due to not finding a valid random boundary map.
- Sphere (S): Using the "Tetrahedron" boundary mapping as a first step, we project all boundary vertices onto the circumsphere to obtain a spherical boundary map. From our experience, this is a safer way to generate a valid spherical boundary map than, e.g., Laplacian smoothing based approaches, which tend to yield some degenerate boundary faces.
- Random Star (*RS*): Finally, by randomly radially displacing the boundary vertices after mapping them to a sphere, we obtain a boundary that is guaranteed to be star-shaped by design. We pick displacement factors uniformly in the range $[1, 10]$. This creates very challenging test cases with domain shapes containing high frequency details.

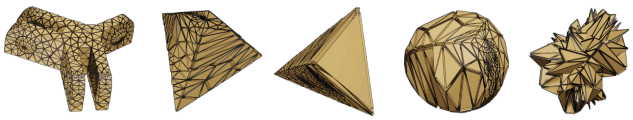

Fig. 13. Each mesh of TetWild (left) has its boundary mapped to four different targets: a Sphere, a "Stiff" Tetrahedron, a Tetrahedron, and a Random Star. The Stiff Tetrahedron and the Random Star are the most challenging cases, but for different reasons, as discussed in Section 6.1.

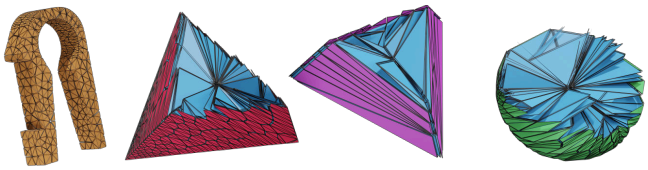

Fig. 14. One mesh of our dataset (left) and three bijective maps to (in this order) tetrahedral, "stiff" tetrahedral and sphere boundaries. None of our maps contain any inverted or degenerate tetrahedra, but they tend to concentrate cells towards the center of the domain.

For all those boundaries (which are summarized in Figure 13), we pick the origin as center; it is inside the kernel by design. To provide an easier starting point for the optimization based methods TLC and FoF, we compute an initial discrete harmonic mapping of the interior, given the fixed boundary map.

6.2 Algorithm Analysis

To keep our full benchmarks within reasonable completion time, we put a per-mesh time limit of 12h on our method (as well as on all alternative methods in the following).

Running our method on the dataset shows that it never produces a non-bijective map but reaches the time limit in ~23.38% of cases. Interestingly, output maps tend to form mesh images that are somewhat compressed near the initial position of the main cluster, as suggested by Figure 14. Figure 15 summarizes the timings we

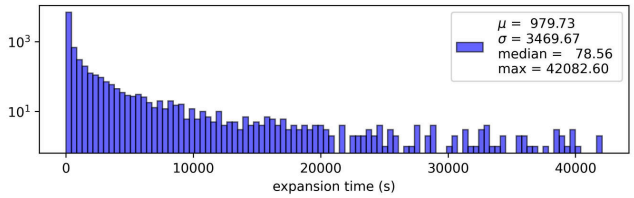

Fig. 15. Distribution of completion timings for all success cases of SaE (log scale). We grouped all 4 target boundary types, as the results are very similar across types, highlighting an interesting property or our method.

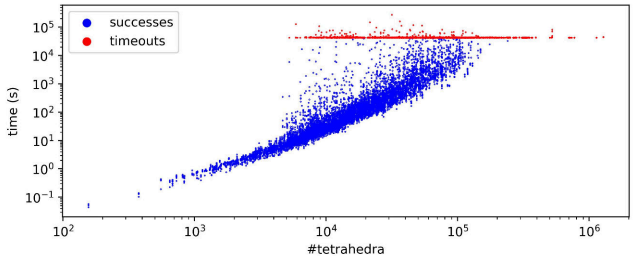

Fig. 16. Runtime depending on the input mesh size. It suggests an exponential relationship between the size of a mesh (initial number of tetrahedra) and the total time required to obtain a valid bijective map. While meshes of a broad range of sizes time out (from around 10^4 tetrahedra and up), a large portion of meshes can be mapped within minutes. Remark: some instances going beyond the 12h limit are due to the granularity of the timeout check in the implementation (e.g. not inside a linear program solve).

obtained for the whole dataset with the four different boundary mappings. It shows that while the majority of meshes can be fully mapped within minutes, a considerable number of meshes can take hours to be mapped. As shown in Figure 16, the mesh size (number of tetrahedra) has a significant impact on the completion time, but is not the sole factor. Figure 16 additionally suggests that overall there is a non-polynomial relationship between the mesh size and the total processing time.

The method's relatively high success rate (within the time limit), as analyzed further in Section 6.3, however, comes at the cost of sometimes requiring some heavy refinement, as summarized by Figure 17. It is also interesting to note that contrary to the termination time, there seems to be no relationship between mesh size and its final growth ratio due to refinement, as shown by Figure 18.

For an overlook of how meshes are expanded in detail, in terms of which expansion operations are used at what frequency, see Figure 19. Clearly, the vast majority of expanded subclusters are trivial and require no star-shapification. Among the the remaining expansions, trivial subclusters with star-shapification tend to be dominant over non-trivial subcluster expansion. Star-shapification for non-trivial subclusters is required very rarely. Figure 20 shows a few examples of how the run time can be distributed between our expansion operations. Notice the high variability regarding which parts of the algorithm are dominant.

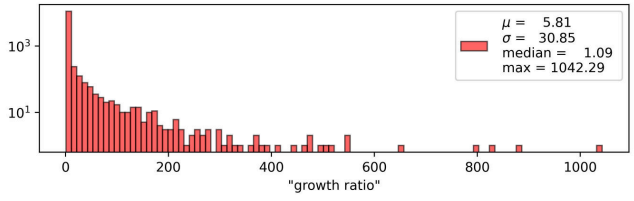

Fig. 17. A distribution showing how much our method tends to make a mesh grow in the number of vertices (log scale). The majority of meshes increase by less than 9%, and about 85% of meshes increase by less than 2×. However, a small portion of the dataset requires heavy refinement, reaching up to 1042 times the initial number of vertices.

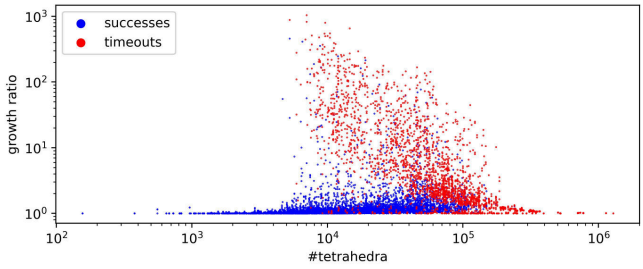

Fig. 18. As this scatter plot suggests, there seems to be no particular relation between the size of a mesh and its growth ratio.

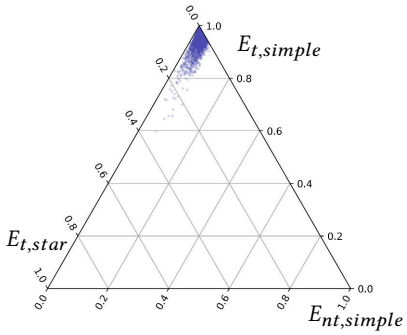

Fig. 19. Each mesh mapped to a point based on the ratio of its vertices expanded by simple expansion of trivial subclusters ($E_{t,simple}$), star-shapifications of trivial subclusters ($E_{t,star}$), and simple expansions of non-trivial subclusters ($E_{nt,simple}$). Cluster star-shapifications (generally <0.3% of all expansions) are omitted.

Regarding vertex relaxation, as discussed in Section 5.5.2, while we observe that it reduces runtime overall, note that it is not a necessary component. There are numerous examples of instances that

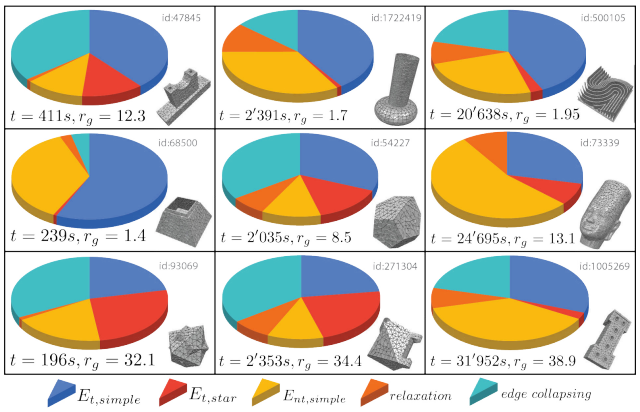

Fig. 20. Timing distribution of the different algorithm parts, for 9 example meshes selected to cover a wide range of total expansion time (t) and refinement ratio (r_g). The parts are: simple expansion of trivial clusters ($E_{t,simple}$), trivial subcluster star-shapifications ($E_{t,star}$), simple expansions of non-trivial subcluster ($E_{nt,simple}$), vertex relaxation, and edge collapsing.

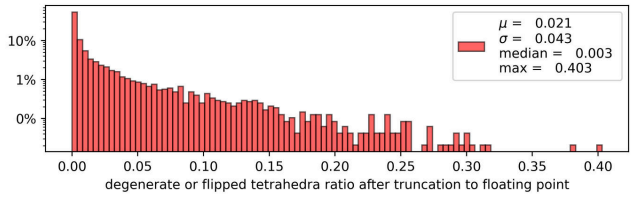

Fig. 21. Truncating the final maps to double precision leads to tiny bijectivity violations for about 48% of them. This histogram shows how far these cases are from being bijective.

terminate within the time limit regardless of whether the relaxation passes are applied or not.

6.2.1 Floating Point Representation. As discussed in Section 5.5, our implementation makes use of exact rational arithmetic. In practice, conversion of the output to a standard floating-point representation may be desired depending on the application context. A naive truncation of the output map to standard double precision preserves bijectivity in ~52% of the cases. Figure 21 details to what extent bijectivity gets violated in the remaining cases due to tiny degeneracies or inversions caused by numerical imprecision. The exploration of means to improve on this aspect—either by performing the conversion more carefully than by naive truncation or by already in the expansion process avoiding near-degenerate configurations whenever possible—is a worthwhile direction for future work.

6.3 Comparison

No previous method is formulated for exactly the same setting as ours (ball-topology input, star-shaped target domain, prescribed boundary map) and has the same sole goal (bijectivity). Nevertheless, comparisons to related methods with at least similar settings and goals can provide valuable insights, when keeping the differences in mind.

Numerical Methods. Considering as success the generation of a map that is bijective within a time limit of 12h, it turns out that our method is significantly more successful than the 3D variants of recent numerical optimization based methods TLC [Du et al. 2020] and FoF [Garanzha et al. 2021]. Both methods' implementations were taken from the corresponding paper authors' online repositories; no parameter tweaking was performed. Success rates are detailed in Table 1 with a focus on the different types of boundary conditions, and in Table 2 relative to the size of the input mesh. Figure 22 furthermore shows the relative distribution of successes across methods. For details about how far these two methods are from succeeding in cases of failure, refer to Figure 23. Of course, these observations are valid only for our setting of ball-topology meshes and the star-shaped domains. TLC and FoF are more general and support a larger class of problems (arbitrary topology, arbitrary domain shape), to which our method could not even be applied. Also, both methods do not perform any mesh refinement, while for our method the amount of refinement can be pretty high in some instances. However, we notice that the amount of refinement performed by our method *in those instances where TLC and FoF succeed* is actually quite low, as shown in Figure 24.

Table 1. Overall success rates (within 12h per mesh) of each method, total and per type of boundary map. Note that while for our method a failure case is simply a timeout, for TLC and FoF it includes timeouts as well as non-bijective output. Our method performs significantly better, especially for the more challenging boundary mappings, namely ST and RS. Additionally, the success rates of our method are very close between the 4 types, highlighting an interesting property of our method. It is worth pointing out that the relatively low success rates of TLC and FoF are not primarily caused by the time limit; even when granting them 1 week per mesh, the total success rates only increase to around 33% and 17%, respectively.

	Total	T	ST	S	RS
TLC	25.07%	47.02%	20.94%	31.45%	0.97%
FoF	12.68%	15.48%	5.31%	22.58%	7.39%
SaE (ours)	**76.62%**	**77.04%**	**77.34%**	**76.41%**	**75.97%**

Table 2. Success rates accumulated over the four boundary types, sorted by mesh size. Our method's success rate is higher for all batches.

batch	#cells range		TLC	FoF	SaE (ours)
0	[156 ,	5125]	32.58%	27.54%	**100.00%**
1	[5125 ,	7516]	48.19%	32.49%	**98.49%**
2	[7516 ,	10490]	39.63%	23.43%	**92.02%**
3	[10490 ,	13913]	34.17%	18.05%	**86.48%**
4	[13930 ,	18934]	29.55%	11.67%	**85.47%**
5	[18934 ,	25559]	22.08%	5.37%	**89.08%**
6	[25559 ,	36373]	15.11%	2.52%	**80.69%**
7	[36373 ,	53548]	11.17%	1.34%	**70.78%**
8	[53548 ,	86404]	10.41%	2.85%	**46.43%**
9	[86404 ,	1283106]	07.73%	1.51%	**16.72%**

Besides robustness and growth, map quality in terms of distortion may be a relevant point of comparison—noting that our method solely focuses on bijectivity and makes no direct effort to minimize distortion. We consider a conformal distortion metric $\mathrm{tr}(J_t^T J_t)/(\det J_t)^{2/3}$ and a volume distortion metric $\det J_t + (\det J_t)^{-1}$ (with J_t being the Jacobian of the linear map of tetrahedron t). This distortion analysis clearly shows that our method produces maps

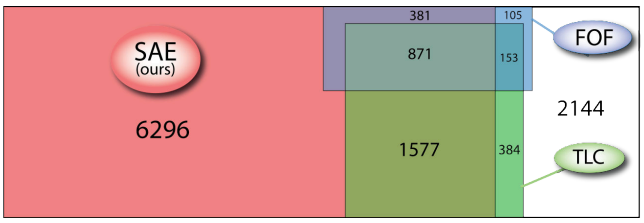

Fig. 22. A roughly area-proportional Venn diagram showing the distribution of success cases by method. Our method (SaE) can map a large number of meshes that other methods cannot (6296), while a smaller number of meshes (642) can be mapped by others but not by ours within the time limit.

Table 3. Summary of the statistics of the various methods. The columns report the success rate (terminating with a map that is bijective in ≤12h), the timeout rate, the runtime normalized by the number of tetrahedra in the input mesh, conformal distortion d_{conf}, volume distortion d_{conf}, and growth ratio. Note that the gap between success rate and timeout rate is due to a method terminating within 12h but outputting a non-bijective map. As expected, compared to the optimization based methods, our method produces much lower quality maps at a lower speed and at the cost of mesh refinement. By contrast, our method shows a significantly higher success rate (in the time limit) than the other state-of-the-art methods, in line with the central goal of our novel framework. For comparability, the runtime, distortion, and refinement numbers refer to the set of instances that are common successes of all three methods.

	success	timeout	runtime per input tet (s)			d_{conf}		d_{vol}		refinement ratio	
		> 12h	min	avg	max	avg	max	avg	max	avg	max
TLC	25.07%	30.72%	**1.03 10^{-4}**	**9.59 10^{-4}**	**5.84 10^{-3}**	**1.78 10^{1}**	**1.20 10^{4}**	3.04 10^{14}	7.03 10^{18}	**1**	**1**
FoF	12.68%	39.24%	3.79 10^{-6}	4.72 10^{-2}	2.52 10^{-1}	1.26 10^{2}	4.93 10^{5}	**4.17 10^{13}**	**1.65 10^{17}**	**1**	**1**
SaE (ours)	**76.62%**	23.38%	2.82 10^{-4}	1.20 10^{-2}	2.55 10^{0}	1.23 10^{203}	2.04 10^{210}		> 10^{308}	1.10	16.63

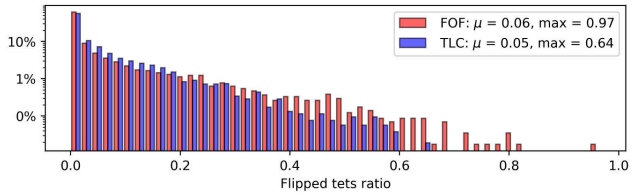

Fig. 23. A histogram of the ratio of flipped tetrahedra in resulting maps whenever TLC or FoF output a non-bijective map. While the average is quite low, both methods produce also results that are quite far from being bijective.

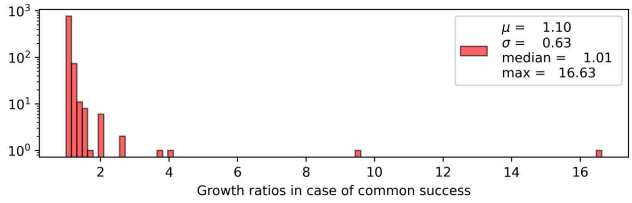

Fig. 24. Growth ratios for meshes for which FoF and TLC also succeed in producing a valid map. Since TLC and FoF both operate without any refinement, this plot gives an indication of how much truly *unnecessary* refinement our method induces. The vast majority of instances cause very little refinement, while a few cause a lot of refinement.

with very high distortion, as shown by Figure 25. Table 3 summarizes the relative statistics of these methods in comparison.

Combinatorial Method. The method FOL [Campen et al. 2016], based on a combinatorial construction, comes with guaranteed bijectivity. We thus focus on comparisons of runtime and the amount of mesh refinement performed to yield a piecewise linear output map. As this method does not support arbitrary convex or star-shaped domain shapes like our method, we restrict this comparison to type S, the sphere domain subset of our dataset. FOL proceeds in two stages, the construction of a bijective map and the generation of a piecewise linear representation of this map. The latter stage, which similar to our method makes use of refinement of the input

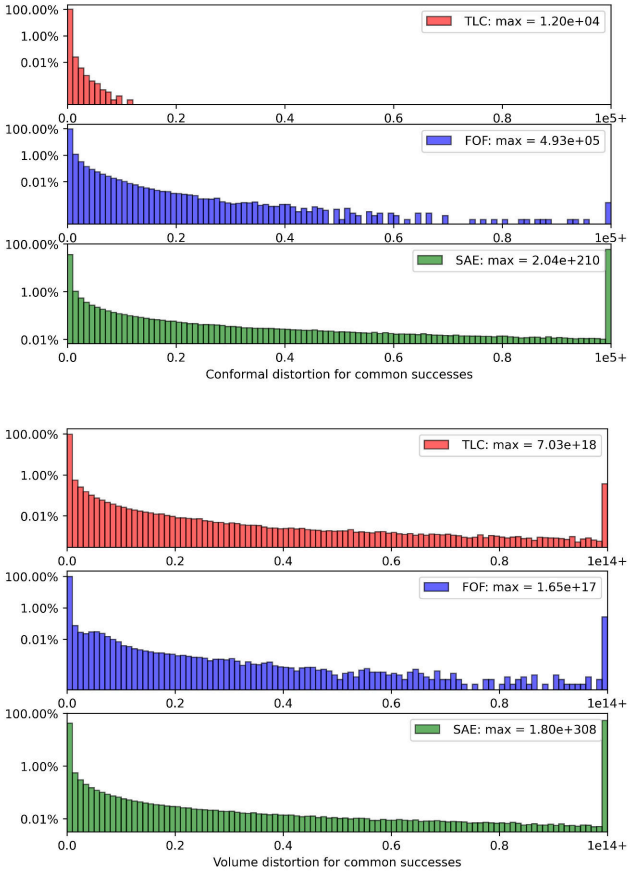

Fig. 25. These histograms show, per method, the conformal distortion and volume distortion of the resulting maps over all tetrahedra of all meshes for which the three methods succeed. Cases where the distortion exceeds 10^5 or 10^9, respectively, are aggregated in the last bar for readability. Not surprisingly, as our method does not target distortion minimization but solely bijectivity, the vast majority of elements generated by our method have a very high distortion in comparison to TLC and FoF. Note that in the case of volume distortion even the maximum representable by double precision is reached—but bijectivity was verified in exact arithmetic, i.e. no element is actually degenerated or inverted by the map.

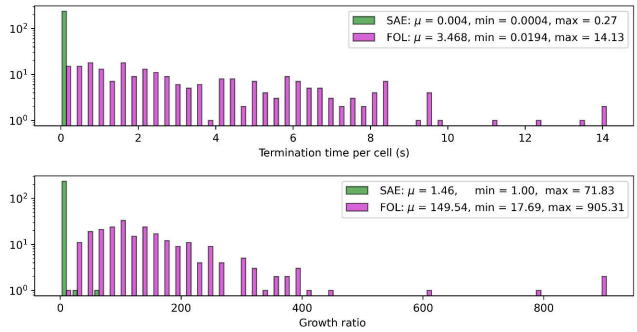

Fig. 26. Comparing the total runtime (per input tet) and the mesh refinement ratio of our method and FOL on the sphere domain subset of the dataset (and restricted to meshes where both methods finished within the time limit). In terms of absolute growth (additional number of cells due to refinement), FOL is at 4.3×10^5 on average while ours is at 3.10×10^3.

mesh, is optional—but needed when such a standard representation is desired. The first stage is generally very fast in comparison, succeeding on each case of our entire dataset in under 3 minutes, and in 4.2 seconds on average. The second stage, however, takes significantly longer than our method and causes significantly more refinement. This is detailed in Figure 26.

6.4 Limitations & Future Work

With the above discussed downsides (exponential runtime behavior, over-refinement, high distortion) in mind, let us now discuss how our method could be improved. In particular, while its robustness is backed theoretically (assuming the condition described in Section 4.3.1 is not violated) and by empirical results, its success within *practically reasonable time* is not guaranteed.

Considering the results and their discussion in the previous section, it becomes clear that the key to obtaining faster results lies in reducing the number of splits necessary to obtain a valid map. Hence, for our approach to reach its full potential, the most important direction for future work is to find alternative yet reliable ways to make expansion cones star-shaped at a lesser cost in terms of refinement. Note in particular that our current implementation performs splits whenever vertices need to be moved in the star-shapification process. Forms of adaptive refinement, coming into play only when strictly necessary, appear attractive. Also, more intelligent forms of vertex relaxation (cf. Section 5.5.2), promoting well-shaped expansion cones and reducing numerical precision demands, are imaginable. The order of expansion, i.e. the order of choosing subclusters (cf. Section 5.6), likewise has an effect and should be investigated further.

Another approach with great potential would be to solve the problem in a "divide-and-conquer" manner. Instead of favoring small subclusters, expanding vertices mostly one by one, recursively splitting clusters into two subclusters of roughly equal size each time may offer benefits overall. This will require a different algorithm to select an expandable subcluster of size around $\frac{k}{2}$ from a k-cluster, for instance by focusing on finding a *sheet* of interior faces that split a cluster into two ball-topology halves.

Finally, while initially shrinking the mesh's interior to a single point guarantees a starting point without any inversions, it is not necessary to go that far. Finding a method to shrink the interior in a smarter, adaptive way might allow for a huge speedup, especially for bigger meshes. One could focus on taking an initial interior map with inversions (coming from some other method) and then shrinking only subsets of interior vertices to multiple local cluster, just enough to turn all inverted elements into degenerate ones. This would already provide a valid starting point for our method.

7 CONCLUSION

In this paper, we introduced a novel framework called *Shrink-and-Expand* to find a bijective map from a ball-topology tetrahedral mesh to a star-shaped target domain, given a boundary map. Aside from a theoretical limitation, namely its general success depending on a hard to verify condition, the current specialization into the Prioritizing Expansion algorithm comes with practical limitations in terms of efficiency.

Benchmarked on a challenging dataset, it proved to sometimes yield impractical run times, numerically challenging map distortions, and high amounts of mesh refinement. Additionally, its implementation is relatively intricate, in particular the star-shapification algorithm. While this may constitute an impediment to immediate practical utility, we also reiterate that we consider the framework a stepping stone for future variants of the method.

Nevertheless, our benchmark provides empirical evidence that, depending on the scenario, even the current version can already offer major benefits compared to state-of-the-art alternatives, in terms of run time as well as time-budgeted success rates, especially for challenging cases with difficult boundary constraints.

ACKNOWLEDGMENTS

D. Bommes has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (AlgoHex, grant agreement No 853343). M. Campen has received funding by the Deutsche Forschungsgemeinschaft (DFG) - 497335132; 456666331.

REFERENCES

Karim Adiprasito and Bruno Benedetti. 2020. Barycentric Subdivisions of Convex Complexes Are Collapsible. *Discrete Comput. Geom.* 64, 3 (2020), 608–626.
Noam Aigerman and Yaron Lipman. 2013. Injective and Bounded Distortion Mappings in 3D. *ACM Trans. Graph.* 32, 4 (2013).
Noam Aigerman and Yaron Lipman. 2015. Orbifold Tutte Embeddings. *ACM Trans. Graph.* 34, 6 (2015).
Marc Alexa. 2023. Tutte Embeddings of Tetrahedral Meshes. *Discrete & Computational Geometry* (2023).
Anthony Anderson, Xiaoming Zheng, and Vittorio Cristini. 2005. Adaptive unstructured volume remeshing – I: The method. *J. Comput. Phys.* 208, 2 (2005), 616–625.
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), 1–12.
Marcel Campen, Ryan Capouellez, Hanxiao Shen, Leyi Zhu, Daniele Panozzo, and Denis Zorin. 2021. Efficient and robust discrete conformal equivalence with boundary. *ACM Trans. Graph.* 40, 6 (2021), 1–16.
Marcel Campen, Cláudio T. Silva, and Denis Zorin. 2016. Bijective Maps from Simplicial Foliations. *ACM Trans. Graph.* 35, 4 (2016).
Tamal Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry Nekhayev. 1999. Topology Preserving Edge Contraction. *Publications de l'Institut Mathématique* 66 (1999), 23–45.

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).
Andreas Fabri and Sylvain Pion. 2009. CGAL: The Computational Geometry Algorithms Library. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '09)*.
Michael Floater and Valérie Pham-Trong. 2006. Convex combination maps over triangulations, tilings, and tetrahedral meshes. *Adv. Comput. Math.* 25 (2006), 347–356.
Michael S. Floater. 1997. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design* 14, 3 (1997), 231–250.
Michael S. Floater. 2003. One-to-One Piecewise Linear Mappings over Triangulations. *Math. Comput.* 72, 242 (2003), 685–696.
Xiao-Ming Fu, Yang Liu, and Baining Guo. 2015. Computing Locally Injective Mappings by Advanced MIPS. *ACM Trans. Graph.* 34, 4 (2015).
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).
Mark Gillespie, Boris Springborn, and Keenan Crane. 2021. Discrete conformal equivalence of polyhedral surfaces. *ACM Trans. Graph.* 40, 4 (2021).
G. Hansen, Irmina Herburt, H. Martini, and M. Moszyńska. 2020. Starshaped sets. *Aequationes mathematicae* 94 (2020).
Steffen Hinderink and Marcel Campen. 2023. Galaxy Maps: Localized Foliations for Bijective Volumetric Mapping. *ACM Trans. Graph.* 42, 4 (2023).
K. Hormann and G. Greiner. 2000. MIPS: An Efficient Global Parametrization Method. In *Curve and Surface Design 1999*. 153–162.
Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4 (2018).
Y. Jin, J. Huang, and R. Tong. 2014. Remeshing-Assisted Optimization for Locally Injective Mappings. *Computer Graphics Forum* 33, 5 (2014).
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).
Tong-Yee Lee, Shao-Wei Yen, and I-Cheng Yeh. 2008. Texture mapping with hard constraints using warping scheme. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (2008), 382–395.
Wentao Liao, Renjie Chen, Yuchen Hua, Ligang Liu, and Ofir Weber. 2021. Real-Time Locally Injective Volumetric Deformation. *ACM Trans. Graph.* 40, 4 (2021).
Yaron Lipman. 2012. Bounded distortion mapping spaces for triangular meshes. *ACM Trans. Graph.* 31, 4 (2012), 1–13.
Marco Livesu. 2020. A Mesh Generation Perspective on Robust Mappings. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, Silvia Biasotti, Ruggero Pintus, and Stefano Berretti (Eds.).
Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Trans. Graph.* 31, 6 (2012).
Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable locally injective mappings. *ACM Trans. Graph.* 36, 4 (2017).
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.
Christian Schüller, Ladislav Kavan, Daniele Panozzo, and Olga Sorkine-Hornung. 2013. Locally Injective Mappings. In *Proc. SGP '13*. 125–135.
Alla Sheffer, Emil Praun, and Kenneth Rose. 2006. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision* 2, 2 (2006), 105–171.
Hanxiao Shen, Zhongshi Jiang, Denis Zorin, and Daniele Panozzo. 2019. Progressive Embedding. *ACM Trans. Graph.* 38, 4 (2019).
Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4 (2015), 70:1–70:9.
Jian-Ping Su, Xiao-Ming Fu, and Ligang Liu. 2019. Practical foldover-free volumetric mapping construction. *Computer Graphics Forum* 38, 7 (2019), 287–297.
William T. Tutte. 1963. How to Draw a Graph. *Proceedings of The London Mathematical Society* 13 (1963), 743–767.
Ofir Weber and Denis Zorin. 2014. Locally Injective Parametrization with Arbitrary Fixed Boundaries. *ACM Trans. Graph.* 33, 4 (2014).
Martin Wicke, Daniel Ritchie, Bryan M Klingner, Sebastian Burke, Jonathan R Shewchuk, and James F O'Brien. 2010. Dynamic local remeshing for elastoplastic simulation. *ACM Trans. Graph.* 29, 4 (2010), 1–11.
Jiazhi Xia, Ying He, Shuchu Han, Chi-Wing Fu, Feng Luo, and Xianfeng Gu. 2010. Parameterization of Star-Shaped Volumes Using Green's Functions. In *Advances in Geometric Modeling and Processing*. Springer, Berlin, Heidelberg, 219–235.

A SUBCLUSTER EXPANDABILITY CHECK

For a given connected subcluster S, we perform a series of checks to determine whether it is topologically-expandable, ordered from least costly, necessary conditions, to more costly, ultimately sufficient conditions. S is topologically-expandable if all of these conditions are met:

(1) $EC(S)$ contains at least one tetrahedron.
(2) The Euler characteristic of $EC(S)$ is 1.
(3) The Euler characteristic of the base of $EC(S)$ is 1.
(4) The Euler characteristic of the mesh spanned by the tips is 1.
(5) All base vertices are 2-manifold within the base.
(6) The base is a single connected component.
(7) All faces of $EC(S)$ are incident to at least one tetrahedron.
(8) All tips are 3-manifold within $EC(S)$.

B STAR-SHAPIFICATION ALGORITHMS

Here we go more in-depth into the implementation of the expansion cone star-shapification, first for trivial subclusters (Algorithm 2), then for nontrivial subclusters (Algorithm 6). Note that all subroutines operate on the same refinement map ψ (thus the same refined mesh M') and the same map ϕ from the vertices of M' to their image position in the domain. Those will be updated by the different parts of the algorithms, in particular each split operation consists of updating ψ.

Algorithm 2 Star-shapification of the expansion cone of a vertex t

Input:
t, a vertex forming a topologically-expandable subcluster $\{t\}$
Output:
ϕ, updated such that $EC(t)$ is star-shaped
ψ, updated to reflect all splits performed

1: **procedure** StarShapify(t)
2: **if** {t} is simply-expandable **then return**
3: **end if**
4: $w \leftarrow$ base vertex of $EC(t)$ with highest valence within the base
5: $SS \leftarrow$ ShellingSequence(t, w) (Algorithm 3)
// Refines the base of $EC(t)$ if necessary, returns a suitable order
6: $(M_S, L_C) \leftarrow$ CollapseSequence(t, SS) (Algorithm 4)
// Moves all cone base vertices (except w and its 1-ring) onto $\partial N_1(w)$, without inverting any tetrahedron. $EC(t)$ now is star-shaped but partially degenerate.
7: ContractionSequence(t, M_S, L_C) (Algorithm 5)
// Now $EC(t)$ is star-shaped.
8: **end procedure**

B.1 Shelling Sequence

As outlined in Section 5.3, the first step of the star-shapification approach is to determine a sequence, called shelling-sequence (SS), in which base vertices can be incrementally collapsed outside-in. We need to maintain ball-topology of the expansion cone throughout for the process to work.

To perform such a collapse, we need to move a vertex at the boundary of the base inwards in such a way that all incident spoke tetrahedra degenerate. This is only possible if the vertex is incident to at most 2 spoke tetrahedra (at the time of collapsing it) as demonstrated by Figure 27. To describe the process more easily in the

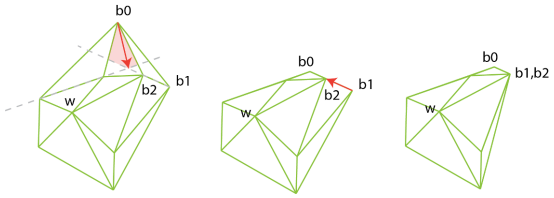

Fig. 27. For the cone base above, the first vertex b_0 of the SS $(b_0 \rightarrow b_1)$ has three incident cells. As such, it is impossible to find a position for b_0 that would make its three incident cells (here projected to faces) degenerate. At best, we could move b_0 to the boundary of the red area, only making two of its incident cells degenerate. Additionally, the next vertex b_1 is now limited to moving exactly towards b_0 (to keep (b_0, b_1, b_2) degenerate), meaning that its incident cells cannot be degenerated either. Note that while this is not an issue in itself, it does not fit the assumptions of our specific star-shapification method.

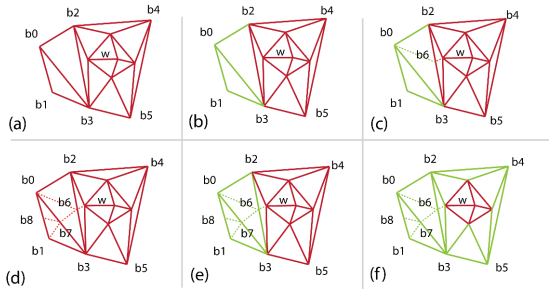

Fig. 28. A cone base during shelling sequence (SS) determination. The red edges are part of the trimmed copy and the green ones are part of the expansion cone but were removed already. (a) The initial expansion cone. (b) The trimmed copy after removing the first two vertices $(b_1 \rightarrow b_0)$. Now we are stuck because all remaining vertices are incident to more than 2 spoke tetrahedra in the trimmed copy. (c) Hence a border edge is split, here edge (b_2, b_3). (d) This split is then back-propagated through what was already removed, inducing two more splits on the "coincident" edges. The trimmed copy is reset to the entire cone. (e) The SS is now $(b_1 \rightarrow b_8 \rightarrow b_0 \rightarrow b_7 \rightarrow b_6)$ and vertex b_2 is now only incident to 2 spoke tetrahedro, i.e. we can continue. (f) The SS is complete as all vertices but those in $N_1(w)$ are removed.

following, we imagine that collapsed vertices are actually removed from the expansion cone, as are their incident edges, triangles, and tetrahedra. The current remainder of the expansion cone we refer to as *trimmed copy* in the course of the process. The boundary of the base in this trimmed copy is referred to as *border*. In essence, border vertices in this trimmed copy need to be removed until only the 1-ring neighborhood of the picked witness vertex w remains. Algorithm 2 uses the simple heuristic of picking the highest valence base vertex as w.

In the rare case that all border vertices in the trimmed copy have more than 2 incident spoke tetrahedra, we split a border edge (not entirely contained in $N_1(w)$), yielding a mid-vertex that can be removed next (it has 2 incident spoke tetrahedra by construction); at the same time, this lowers the numbers of incident spoke tetrahedra of the adjacent border vertices by 1 each. This can be repeated until their valence is 2. This ensures we can proceed.

Algorithm 3 Finding a shelling sequence for an expansion cone

Input:
t, the tip vertex to expand
w, the desired witness vertex of $EC(t)$
Output: SS, a viable shelling sequence

procedure ShellingSequence(t, w)
  $SS \leftarrow \emptyset$
  $SS_b \leftarrow \emptyset$
  $Trim \leftarrow EC(t)$ // trimmed copy.
  $B \leftarrow EC(t) \setminus N_1(w)$ // base vertices that are not neighbors of w.
  $Mid \leftarrow \emptyset$ // list of mid-vertices created by split back-propagation.
  $C \leftarrow \emptyset$ // list of *candidates* to remove. Used to re-do SS after splits.
  repeat
    if $C = \emptyset$ **then**
      if $Mid \neq \emptyset$ **then**
        $C \leftarrow SS_b \cup Mid$
        $Mid \leftarrow \emptyset$
      else
        $C \leftarrow B \cap Trim$
      end if
    end if
    Let $b \in C$ be the lowest cell-valence vertex removable from $Trim$ while maintaining ball-topology
    if b has cell-valence ≤ 2 **then**
      Append b to SS
      Remove b from $Trim$ and from C
    else
      Find an edge e on the boundary of the base of $Trim$ that is connected to at least one vertex that is not part of $N_1(w)$
      Split e
      Back-propagate this split along the dual path on the base of $EC(t)$ from e to the boundary of the base, following the current SS. Gather all mid-vertices from these splits in Mid
      $SS_b \leftarrow SS$
      $SS \leftarrow \emptyset$
      $Trim \leftarrow EC(t)$
      $C \leftarrow \emptyset$
    end if
  until $Trim = N_1((w, t))$
end procedure

Of course, splitting a border edge might increase the valence of a vertex in the already removed part above 2. We thus propagate the split through the part of the base that is not in the trimmed copy anymore, splitting all edges that (in the collapsed state) contain the split point. The SS-so-far is then recomputed on the refined base, removing vertices in the same order as before but interleaving the removal of a newly introduced split vertex whenever possible. See Figure 28 for an example of an SS determination, and Algorithm 3 for the full process.

B.2 Collapse Sequence

Given a shelling sequence, and assuming the mesh was suitably refined through edge splits as described in Section B.1 to make this sequence viable, we can now actually modify the map ϕ to execute the collapses, in order to make the expansion cone star-shaped. We will use the trimmed copy as a helper again. Additionally, to simplify writing, we will denote the witness vertex as w and the tip vertex

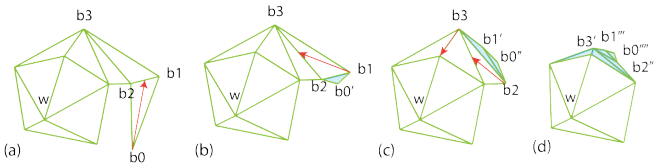

Fig. 29. The collapse sequence corresponding to the SS ($b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3$), shown on a cone base. As usual, the orange arrows show the mid-vertices' movement and blue triangles are actually degenerate. (*a*) We first collapse b_0 to the face of its incident tet that is not incident to b_0, i.e. (b_1, b_2, t), with t being the tip vertex (not shown here). (*b*) When the spoke (b_1, t) is split and its mid-vertex b_1' is moved to the barycenter of face (b_2, b_3, t), the previously collapsed vertex b_0' is "dragged along" by splitting its spoke (b_0', t) and moving the mid-vertex b_0'' to the barycenter of face (b_1', b_2, t). (*c*) This is repeated for b_2, and the mid-vertex b_3' is moved to the mid-point of the edge (b_4, t), which is the common edge of its two non-degenerate incident cells. (*d*) As a result of the collapse sequence, the expansion cone is star-shaped, with all cells non-incident to the witness vertex w and t being collapsed to the boundary of $N_1((w, t))$. All degenerate cells will be expanded by the contraction sequence (see Section B.3).

as t. We encourage the reader to first look at Figure 29 to get a sense of how a collapse sequence works, with a small but complete example. For each base vertex b of the sequence, we distinguish two cases, depending on the number of incident cells at the moment of its deletion from the trimmed copy. If it has only a single incident cell, we split the spoke (t, b) and move the mid-vertex b' to the barycenter of the one face of that cell that is not incident to b. This incident cell is now degenerate (since we employ an exact number type, as discussed in Section 5.5, this is the case also in practice) and "embedded" inside this target face. This means that we reduced the number of faces defining the geometry of the cone boundary, thus bringing it closer to being star-shaped. Similarly, if there are two cells incident to b at the moment of its removal from the trimmed copy, the spoke (t, b) is split as well, but the mid-vertex b' is moved to the mid-point of the spoke shared by those two cells that is not (t, b). Note that these movements do not cause inversions because the target point is contained in the mid-vertex' expansion cone's kernel in both cases. Whenever a vertex is collapsed, all vertices that were collapsed before to an edge or face that is incident to it are "dragged along", in a recursive manner, such that the boundary is always the same as the trimmed copy's after this vertex was removed from it. Again, spoke edges are split to make sure the movement is possible. Algorithm 4 formally defines this stage. As a reminder, this part of the star-shapification process assumes that all cells inside the cone are initially non-degenerate, as discussed in Section 5.3.

In the end, L_C, the *Collapse List*, contains the list of successive collapses $(b, target)$, where b is the collapsed vertex and *target* is the set of vertices of the simplex (face or edge) onto which b is collapsed. M_S, the *Split Map*, initially maps each vertex to itself and whenever we perform a split $(b, t) \rightarrow b'$, then $M_S(b) = b'$.

B.3 Contraction Sequence

Once the collapse sequence has been executed, we have an expansion cone that is star-shaped but contains degenerate tetrahedra. The third and final stage thus consists of *contracting* the expansion

Algorithm 4 Collapsing expansion cone vertices following the SS to make the cone star-shaped (but partially degenerate).

Input: t, SS,
Output: M_S, L_C, the Split Map and the Collapse List.

procedure CollapseSequence(t, SS)
  $L_C \leftarrow \emptyset$
  $Trim \leftarrow EC(t)$
  $M_S(v) \leftarrow v, \forall v \in EC(t)$
  repeat
    Let b be the front of SS. Remove b from SS.
    Split $(b, t) \rightarrow b'$
    $M_S(b) \leftarrow b'$
  // Collapse b':
    if cell-valence of b in *Trim* is 1 **then**
      Let (t, v_i, v_j, b) be its only incident cell in *Trim*.
      Set $\phi(b')$ as the barycenter of face (t, v_i, v_j).
      Append $(b, (t, v_i, v_j))$ to L_C
    else if cell-valence of b in *Trim* is 2 **then**
      Let (t, v_i, v_k, b) and (t, v_j, v_k, b) be its two incident cells.
      Set $\phi(b')$ as the center of edge (t, v_k).
      Append $(b, (t, v_k))$ to L_C
    end if
  // Back-propagate the collapse:
    for all $(u, \{u_i\})$ in L_C **do**
      if $M_S(u_i) \neq u_i$ for at least one u_i **then**
        Split $(u, t) \rightarrow u'$
        Set $\phi(u')$ as the barycenter of $\{M_S(u_i)\}$
        Replace $(u, \{u_i\})$ with $(u', \{M_S(u_i)\})$ in L_C
      end if
    end for
    Remove b from *Trim*
  until $SS = \emptyset$
end procedure

cone towards the witness vertex, progressively creating space for the tetrahedra that were collapsed to recover a positive volume. We encourage the reader to start with Figure 30 to get an initial grasp of this subtle stage. We start by contracting each vertex b of the *core*, i.e. the set of base neighbors of w, by splitting its spoke edge (b, t)

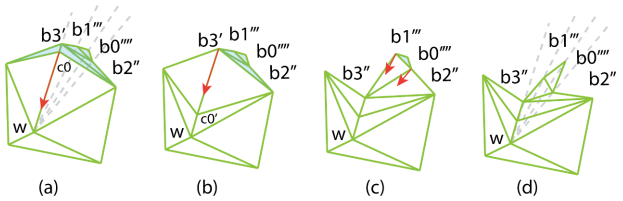

Fig. 30. The contraction sequence following the collapse sequence of Figure 29. The first step is to contract the *core* vertex c_0 of $N_1(w)$, onto which other vertices were collapsed (left). Contracting this vertex makes two cells incident to b_3' non-degenerate, giving room for the mid-vertex b_3'' to contract towards w. Moving b_3'' in turn gives room for the vertices that were collapsed onto faces incident to it. Repeating this in reverse order of the SS successively expands all initially degenerate cells, while maintaining star-shapedness. In the end, each mid-vertex $b_j^{(k)}$ lies on the same triangle (b_j, t, w), as suggested by the gray dashed lines.

Algorithm 5 Contracting a collapsed expansion cone, following the collapse sequence

Input: t, w, M_S, L_C,
procedure ContractionSequence(t, w, M_S, L_C)
  // Core contraction:
  Let $B_C \subset N_1(w) \setminus t$ be the set of base vertices such that face (b, t, w) is not a boundary face of $EC(v)$
  for all $b \in B_C$ **do**
    Split $(b, t) \rightarrow b'$
    Set $\phi(b')$ as the barycenter of (b, t, w)
  end for
  // Full contraction:
  for all collapses $(b, target)$ in L_C, in reverse order **do**
    if $\nexists (c, target_c) \in L_C \,|\, b \in target_c$ **then**
      skip b
    else
      split $(b, t) \rightarrow b'$
      // Contraction position computation:
      if $target = (t, v)$ **then**
        Set $\phi(b')$ as the barycenter of $(b, t, M_S(v))$
      else if $target = (t, v_i, v_j)$ **then**
        Compute the intersection between the (non-face) triangle (b, w, t) and the (face) triangle (v_i, v_j, t). With x and y being the endpoints of this intersection (and either $x = \phi(t)$ or $y = \phi(t)$)
        $\phi(b') \leftarrow (x + y + \phi(b))/3$
      end if
    end if
  end for
end procedure

and moving the mid-vertex b' towards the witness vertex, to the barycenter of the spoke face (b, t, w). This creates space for the last collapsed vertices to move towards the witness vertex, in turn creating space for the previously collapsed vertices. The key here is to always contract towards the witness vertex's spoke edge to ensure that there is always space for the next vertices to be contracted, and that the boundary remains star-shaped. The full contraction process is formally defined by Algorithm 5.

Precision: As in Section 5.5.1, we also here reduce number precision while maintaining correctness. Instead of moving the newly-created vertices exactly towards the (t, w) axis, we only enforce that vertices maintain the proper radial order, as observed at the end of the collapse sequence. This approach proved efficient, but does not entirely guarantee a valid result. Hence, we fall-back to exact contraction in the cases it creates inversions or degeneracies.

C CLUSTER ISOLATION

The formal description of the cluster isolation process is given by Algorithm 8. Section C.1 provides the foundation.

C.1 Split Vertex Expandability Proof

In this section, we provide a proof that a cluster created by performing splits on multiple coincident edges can always be simply-expanded, without further refinement. Let us start with a helpful definition:

Algorithm 6 Star-shapification of the expansion cone of a k-subcluster, with $k > 1$

Input:
  $C = \{t_i\}$, a topologically-expandable subcluster
Output:
  ϕ, updated such that $EC(C)$ is star-shaped
  ψ, updated to reflect all splits performed
procedure StarShapify(C)
  if {t} is simply-expandable **then return**
  end if
  Let $\{b_j\}$ be the base vertices of $EC(C)$
  Let $\{f_b\}$ be the base triangles of $EC(C)$
  // Simulate:
  Extract the base of $EC(C)$ as a separate mesh, denoted EC_s
  Add a new isolated tip vertex t to EC_s
  For all faces f_b of EC_s, add a new cell (f_b, t) to EC_s // EC_s is thus a single-tip version of the original $EC(C)$.
  StarShapify(t) // applied to EC_s
  // Duplicate splits:
  Let $L = \{(a, b) \rightarrow c\}$ be the list of splits performed on EC_s
  Let ϕ_s be the map resulting from star-shapification of EC_s)
  Let $Dupl(v)$ be a (vertex $\rightarrow$ set of vertices) map
  $Dupl(b_j) \leftarrow b_j$ for all base vertices and
  $Dupl(v) \leftarrow \varnothing$ for all other vertices
  $X \leftarrow \varnothing$
  for all split $s \in L$ **do**
    if $s = (b_i, b_j) \rightarrow b_k$ was done on the base **then**
      split (b_i, b_j)
      $\phi(b_k) \leftarrow \phi_s(b_k)$
    else
      Then $s = (b_j^k, t) \rightarrow b_j^{k+1}$, with b_j^k being the vertex obtained after the k-th split of the edge $(b_j, t) \in EC_s(C)$ (and $b_j^0 = b_j$) and with $Dupl(b_j^k) \rightarrow \{b_{j,m}^k\}$
      for all edges $(b_{j,m}^k, t_i) \in M'$ **do**
        split $(b_{j,m}^k, t_i) \rightarrow b_{j,n}^{k+1}$
        add $b_{j,n}^{k+1}$ to $Dupl(b_j^{k+1})$
        // $b_{j,m}^k$ is the m-th duplicate of the vertex obtained by performing the k-th split of an edge connecting the j-th base vertex and one of the tip vertices.
        append $b_{j,n}^{k+1}$ to X
      end for
      Set $\phi(v) = \phi_s(b_j^{k+1}), \forall v \in Dupl(b_j^{k+1})$.
    end if
  end for
  ExpandDuplicates(X) (Algorithm 7)
end procedure

Definition C.1 (Wedge). For an edge $e = (v_a, v_b)$, we define its *wedge* $W(e)$ as the submesh consisting of all non-degenerate cells incident to e. We call it a *proper wedge* if it contains exactly two boundary faces incident to e.

$a, b \in \mathbb{R}^3$ will refer to the positions of v_a and v_b, and $[a : b]$ is the line segment from a to b. We will write $Ker(S)$ for the kernel of a subset $S \subset \mathbb{R}^3$.

Lemma 1. *For an edge $e = (v_a, v_b)$, if wedge $W(e)$ is proper, then it is star-shaped.*

Algorithm 7 Expansion of the duplicate mid-vertices created during nontrivial subcluster star-shapification (Algorithm 6)

Input:
X, the list of duplicate mid-vertices to expand
Output:
ϕ, updated such that vertices in X are expanded
procedure ExpandDuplicates(X)
  $\phi_{\text{init}}(v) \leftarrow \phi(v), \forall v \in X$
  $X' \leftarrow$ randomly ordered list of X
  $\phi(v) \leftarrow \phi_{\text{init}}(v), \forall v \in X$ // reset to initial positions
  repeat
    Pick the first vertex $v \in X'$ that is simply-expandable.
    If there is none, **goto** 3
    Set $\phi(v)$ as the center of the kernel of $N_1(v)$.
    $X' = X' \setminus v$.
  until $X' = \emptyset$
end procedure

Algorithm 8 Isolation of two clusters that were split apart, to ensure that the boundaries of their submeshes contain no degenerate faces

Input:
C_a and C_b, two clusters with at least two edges connecting C_a and C_b
Output:
ϕ, updated such that $\partial Submesh(C_a/C_b)$ contain no degenerate faces
ψ, updated to reflect all splits performed
procedure Isolate(C_a, C_b)
  Let E_{ab} be the set of edges (v_i, v_j) such that $v_i \in C_a$ and $v_j \in C_b$
  repeat
    Pick $e \in E_{ab}$ such that the submesh consisting of all cells incident to e is star-shaped (exists, see Appendix C.1)
    Split $e \rightarrow v$
    Set $\phi(v)$ as a center of $N_1(v)$
    $E_{ab} \leftarrow E_{ab} \backslash e$
  until $E_{ab} = \emptyset$
end procedure

Proof. By construction, $[a : b] \subseteq Ker(W(e))$. Then, the only way for $Ker(W(e)) = [a : b]$ to be true, is for it to be defined by at least three halfspaces intersecting $[a : b]$. However, by construction, $Ker(W(e))$ is defined as the intersection of two boundary faces incident to $[a : b]$ and an arbitrary number of faces f_k. Since all faces f_k are either (v_a, v_i, v_j) or (v_b, v_i, v_j), with $v_i, v_j \neq v_a$ and $v_i, v_j \neq v_b$ and no two vertices can be coincident (by construction), the two boundary faces of $W(e)$ that are incident to e define the only halfspaces intersecting at $[a : b]$ and thus $Ker(W(e)) \neq [a : b]$. Finally, since the intersection of a finite number of halfspaces is either convex or empty, and since $W(e)$ is not empty by construction, there must be a point $x \notin [a : b]$ such that $x \in Ker(W(e))$ and $W(e)$ is thus star-shaped. □

Now consider two distinct clusters C_a and C_b such that $|C_a| > 0$, $|C_b| > 0$ and $|E_{ab}| > 0$ for the set of edges $E_{ab} = \{(v_i, v_j \mid v_i \in C_a, v_j \in C_b\}$. The submesh consisting of the union of 1-ring neighborhoods of all these coincident edges clearly has ball-topology. Now, let us assume that all edges $e_{i,j} = (v_i, v_j) \in E_{ab}$ were split, creating a cluster of mid-vertices $V_{ab} = \{v_{i,j}\}$. Since $N_1(v_{i,j})$ is geometrically and topologically equivalent to $N_1(e_{i,j})$, we can define the wedge $W(v_{i,j})$ of a mid-vertex, for which Lemma 1 holds (by construction).

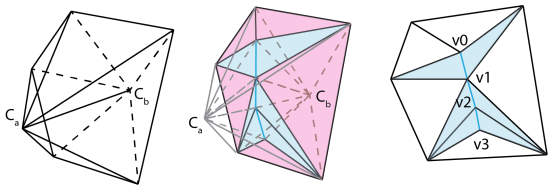

Fig. 31. Left: the union of the 1-ring neighborhoods of all edges connecting clusters C_a and C_b is a ball-topology, star-shaped mesh. Middle: by splitting all cluster-connecting edges, we obtain a disc-topology triangle mesh called the *mid-surface* (blue triangles are degenerate). Right: finding a mid-vertex whose *non-degenerate* 1-ring neighborhood is ball-topology is equivalent to finding an interior vertex of the mid-surface whose non-degenerate 1-ring neighborhood is *disc*-topology. Here, vertices v_0 and v_3 are *Type 2* and vertices v_1 and v_2 are *Type 3*. Our proof is based on showing that there always exists at least one *Type 2* vertex.

Lemma 2. *For a set of edges E_{ab} as defined above, if all edges are split there exists a vertex $v \in V_{ab}$ such that its wedge $W(v)$ is proper.*

Proof. We can transform the problem into a 2D one, by taking $Submesh(C_a) \cap Submesh(C_b)$ and removing all simplices incident to either a vertex of C_a or C_b. The result is a submesh consisting of all simplices incident to at least one vertex of V_{ab}, i.e. a disc-topology surface mesh (see Figure 31). We will call this submesh the *mid-surface* and denote it as Mid_{ab}. By construction, the interior vertices of Mid_{ab} are the simply-connected mid-vertices V_{ab}. We distinguish four types of interior vertices depending on their neighborhood in Mid_{ab}:

Type 0 have only interior neighbors (→ their wedge is empty).
Type 1 have a *single* boundary neighbor (→ their wedge is empty).
Type 2 incident to a single fan of triangles (→ their wedge is proper).
Type 3 incident to multiple, disconnected fans of triangles and/or boundary vertices (→ their wedge is not proper).

We show that there always exists a Type 2 vertex in Mid_{ab}. First, there cannot be only Type 3 vertices, because each of them is connected to multiple disconnected parts of the boundary. This means that there must be at least another boundary vertex between two of its incident components, implying that there must be a Type 1 or Type 2 vertex. Furthermore, since there must be at least one non-degenerate face between two non-degenerate components of a Type 3 vertex, it implies that there must be at least one Type 2 vertex. □

Hence, starting from the initial set of mid-vertices between C_a and C_b, we can find one with a proper wedge and therefore expand it by moving it to a point inside the kernel of the wedge. This operation can then be repeated until all vertices of V_{ab} have been moved.

Note that the 2D expansion problem underlying this argument is conceptually close to the second step of Shen et al. [2019], the difference being that in their case an expansion order is predetermined as a reverse collapse sequence.