A Progressive Embedding Approach to Bijective Tetrahedral Maps driven by Cluster Mesh Topology

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

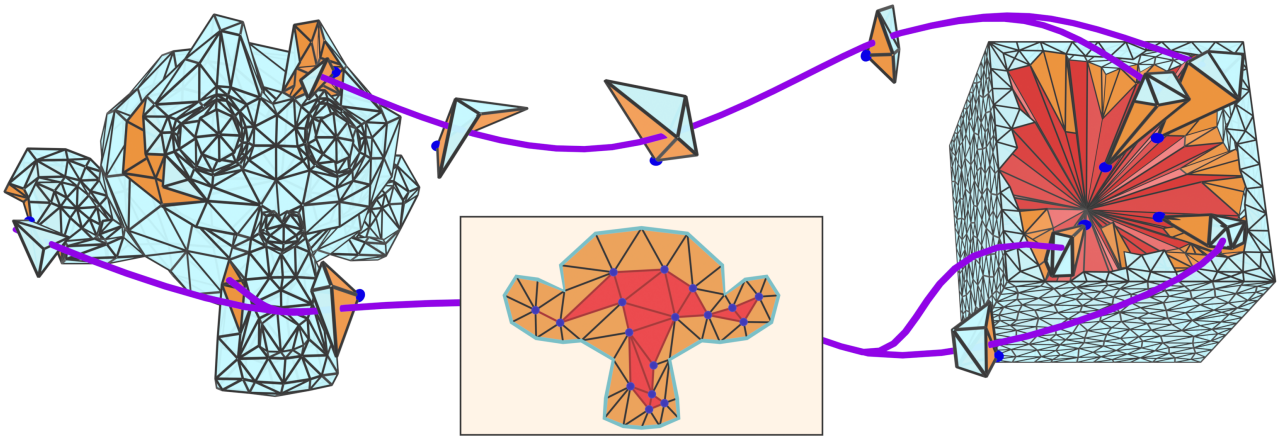


Fig. 1. Given a ball-topology tetrahedral mesh (left) and star-shaped boundary constraints, we generate a bijective piecewise linear map for the interior (right). We improve the recent Shrink-and-Expand approach [Nigolian et al. 2023], which relies on a two-step process. During the first *shrinkage* step, all interior vertices are clustered to a single focal point, degenerating most tetrahedral elements. In the second *expansion* step, interior vertices (blue) are successively detached from the cluster while ensuring that tetrahedra formed by expanded vertices are strictly positively oriented (orange). We introduce the concept of the *cluster mesh* (red), motivated by the observation that its topology provides all information necessary to efficiently determine a viable expansion sequence.

We present a novel algorithm to map ball-topology tetrahedral meshes onto star-shaped domains with guarantees regarding bijectivity. Our algorithm is based on the recently introduced idea of Shrink-and-Expand, where images of interior vertices are initially clustered at one point (Shrink-), before being sequentially moved to non-degenerate positions yielding a bijective map (-and-Expand). In this context, we introduce the concept of the *cluster mesh*, i.e. the unexpanded interior mesh consisting of geometrically degenerate simplices. Using local, per-vertex connectivity information solely from the cluster mesh, we show that a viable expansion sequence guaranteed to produce a bijective map can always be found as long as the mesh is *shellable*. In addition to robustness guarantees for this ubiquitous class of inputs, other practically relevant benefits include improved parsimony and reduced algorithmic complexity. While inheriting some of the worst-case high run time requirements of the state of the art, significant acceleration for the average case is experimentally demonstrated.

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

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.

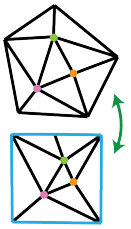
0730-0301/2024/12-ART170
https://doi.org/10.1145/3687992

Additional Key Words and Phrases: tetrahedral mesh, volumetric maps, progressive embeddings, bijection, homeomorphism, star-shaped

ACM Reference Format:
Valentin Z. Nigolian, Marcel Campen, and David Bommes. 2024. A Progressive Embedding Approach to Bijective Tetrahedral Maps driven by Cluster Mesh Topology. *ACM Trans. Graph.* 43, 6, Article 170 (December 2024), 14 pages. https://doi.org/10.1145/3687992

1 INTRODUCTION

In the context of geometry processing, a common problem is to find a bijective *map* between two domains. The resulting pointwise correspondences are beneficial for a variety of applications, including for instance texture mapping [Koniaris et al. 2014], mesh generation [Pietroni et al. 2022] and many more. Specifically important are piecewise linear (PL) maps represented by a tetrahedral mesh equipped with vertex positions for source and target domain. These are bijective if and only if there are no inverted or degenerate tetrahedra, and no self-intersections. The figure on the right illustrates a 2D example, where a pentagonal source domain is bijectively mapped onto a square target domain, defined by the blue prescribed boundary edges. Unfortunately, in three dimensions, generating such bijective maps for tetrahedral meshes of arbitrary topology and subject to general non-convex boundary constraints is still an open problem. In this work, we target the setting of mapping *ball-topology* source



meshes to *star-shaped* target domains, defined by prescribed locations for their boundary vertices. Please note that aside from star-shapedness and non-degeneracy, there are no other requirements on the boundary constraints, particularly no restrictions in terms of distortion. Several key requirements are essential for the practical relevance of a mapping algorithm:

(R1) Robustness. It is essential that for each valid input, consisting of a source tetrahedral mesh and target boundary vertex positions, a valid bijective piecewise linear map is generated. The ultimate goal is provable robustness for a well-defined and broad set of inputs.

(R2) Run Time. Efficiency and scalability are crucial for practical applications. Please note that provable robustness alone is of limited value if the actual wall-clock run time is prohibitively large.

(R3) Parsimony. Depending on the boundary constraints, mesh refinement is sometimes inevitable to warrant the existence of a bijective map. The goal of parsimony consists in minimizing the number of additional vertices, typically also key to bound run time.

(R4) Precision. Maps are often a single component of a larger system, where typically the other components operate with standard floating-point number types. Consequently, a common requirement is the ability to find a map that remains valid after truncating vertex coordinates to IEEE 754 double precision.

(R5) Distortion. For a given source domain and boundary constraints, there exist infinitely many bijective maps. The common goal is to find a low-distortion map w.r.t. some application-dependent distortion metric. Please note that in practice it is often sufficient to find any bijective map, which gives rise to a low-distortion map via subsequent optimization while preserving bijectivity, e.g. through barrier distortion energies combined with line search.

Our progressive embedding approach is based on the idea of *Shrink-and-Expand* (SaE), which has been introduced in [Nigolian et al. 2023]. We will first briefly summarize all important concepts of SaE before explaining the critical limitations of the original algorithm, directly motivating our key improvements.

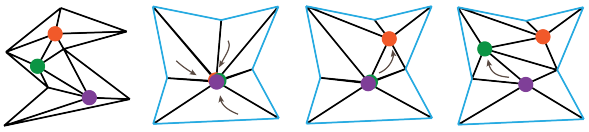


Fig. 2. A 2D overview of the core concept of the SaE framework that our work improves upon. (a) a ball-topology mesh with three interior vertices. (b) boundary vertices' prescribed positions (blue) give a star-shaped target domain. Interior vertices are initially coincident, degenerating some of the elements. In (c) and (d) interior vertices are sequentially detached to positions yielding non-degenerate elements.

Shrink-and-Expand (SaE). Given a ball-topology mesh and non-degenerate, star-shaped prescribed boundary positions, SaE starts with the initialization of positions for interior vertices, allowing *degenerate* tetrahedra due to edges of zero length, but ensuring that there are no inverted tetrahedra. This first part of SaE consists in geometrically *shrinking* interior edges to zero length, however, without making any topological modifications to the mesh. The star-shapedness of the target domain warrants the existence of an interior point p_c that forms positively oriented tetrahedra with all boundary triangles. Consequently, an initialization without inverted tetrahedra is trivially constructed by shrinking all edges between interior vertices to p_c. The second part of SaE consists in sequentially resolving degenerate elements by displacing vertices in order to *expand* their incident tetrahedra without ever creating inversions or novel degeneracies. The expandability of a vertex, i.e. the ability to displace the vertex without creating inverted or degenerate elements, depends on topological as well as geometric conditions, requiring the so-called *expansion cone* to be ball-topology and geometrically star-shaped. Luckily, the geometric conditions can always be satisfied through a robust star-shapification process, however, potentially at the cost of mesh refinement via *edge splits*. In each iteration of the expansion phase, the algorithm first determines a single vertex (or sometimes a subset of vertices) that satisfies the topological conditions, then ensures the geometric conditions, and finally expands some degenerate tetrahedra. Figure 2 gives a brief overview of the core steps of SaE in 2D.

Limitations of [Nigolian et al. 2023] . To obtain robustness in the sense of (R1), SaE depends on a viable expansion sequence, i.e. existence of at least one expandable vertex (or subcluster) in each iteration. While the original SaE approach [Nigolian et al. 2023] demonstrated exceptional robustness for a large and challenging dataset of inputs, the theoretical question of the existence of a viable expansion sequence remained open. Specifically, the existence of such a sequence could not be verified for the 23.38% of inputs where the time limit of 12h was reached. This major computational obstacle results from the combinatorial complexity of potentially exploring all 2^n subsets of vertices of an unexpanded set of n vertices. Moreover, the evaluation revealed cases of suboptimal behavior w.r.t. parsimony (R3), and, precision (R4), often resulting from excessive mesh refinement and precision blow-up of rational numbers.

Contributions. To overcome the theoretical limitations of [Nigolian et al. 2023], we introduce the concept of the *cluster mesh*, consisting of all interior simplices with coincident vertices. In Sec. 3.2, we show that a simple topological condition ensures the expandability of a cluster mesh vertex, which only depends on its 1-ring neighborhood within the cluster mesh. Most importantly, a viable expansion sequence can be guaranteed as long as the tetrahedral mesh is shellable – not even requiring the subcluster expansion of [Nigolian et al. 2023]. For non-shellable meshes, however, our algorithm might fail if the cluster mesh turns into a NOBUENOSS, i.e. a simply connected (non-pure) surface with Euler characteristic 1 that has no boundary, yet does not separate space, cf. Sec. 4.1. Note that such cases are detected on the fly during expansion, without any additional cost. Despite extensive experiments, we did not encounter such a pathological case in practice. We provide a manually designed example in Sec. 4.1, in the hope that, based on our characterization, future work will be able to extend guarantees to arbitrary non-shellable inputs. Another practically relevant effect of the cluster mesh perspective is that the cost of checking expandability is strictly

bounded by the input mesh and thus independent of on-the-fly refinement. The cost solely depends on the size of the 1-ring neighborhood within the cluster mesh, which never grows since the algorithm only performs edge splits outside the cluster mesh. In contrast, the cost of checking expandability with the help of expansion cones, as done in [Nigolian et al. 2023], grows due to mesh refinement and thus might even become a critical bottleneck. While the cluster mesh perspective allows our algorithm to satisfy the requirement of robustness (R1), we additionally demonstrate, in Sec. 5, significant improvements w.r.t. run time (R2), parsimony (R3), and precision (R4). These are related to a novel inflation process that enables a larger number of expansion candidates (Sec. 3.1), an on-the-fly precision truncation heuristic (Sec. 4.2.3), and a Chebyshev-center-driven geometric position optimization (Sec. 4.2.7). Specifically, the performance increase allows our method to reduce the number of maps that cannot be generated within 6 hours to 4.2%, compared to 23.8% of [Nigolian et al. 2023]. The code of our implementation is publicly available at https://www.algohex.eu/publications/cluster-mesh-sae.

2 RELATED WORK

Through the literature, maps are constructed in various ways, generally in terms of what class of objects they can handle, what type of target domain those can be mapped to, and the approach they take to do so. Some techniques are restricted to ball-topology meshes [Campen et al. 2016], while others accept meshes of any genus as input [Du et al. 2020; Garanzha et al. 2021]. Mapping methods can be limited to specific output shapes (e.g. convex), and not all are compatible with per-vertex prescribed positions [Campen et al. 2016]. While some applications of volumetric maps allow some tolerance regarding boundary constraints satisfaction (e.g. deformation or shape correspondence), the satisfaction of exact boundary conditions is an absolute necessity for others, like parameterization or hexahedral meshing [Brückler and Campen 2023; Brückler et al. 2022a,b]. Moreover, meshes of arbitrary genus can be decomposed into ball-topology "blocks", which are individually mapped and then recomposed into a global *atlas*. To enforce boundary correspondence between neighboring blocks in the target domain (which is not guaranteed in general), one can use a *seamless map*. These imply boundary conditions that need to be satisfied exactly, ensuring pointwise consistency on the interfaces between blocks. This is a well-studied problem, with related work in 2D [Campen et al. 2019; Levi 2021; Zhou et al. 2020], as well as in 3D [Liu and Bommes 2023; Nieser et al. 2011]. In an orthogonal yet complementary fashion, one can generate a map between arbitrary shapes by *composing* maps between these shapes and a common target domain [Kanai et al. 1997; Lipman and Funkhouser 2009; Schmidt et al. 2019, 2020; Weber and Zorin 2014]. In this context, robustly generating maps, even in our restricted setting, is an essential component of general tetrahedral maps. We refer to [Cherchi and Livesu 2023] for further review of the numerous applications of volumetric maps, such as solid texturing or volume registration.

2.1 Optimization-Based Tetrahedral Maps

Many mapping techniques involve minimizing per-element distortion, meaning that the elements' shapes should be *as close as possible* in both initial and target domains (R5). This change of shape is evaluated using a so-called *energy function*, which serves as an objective function for optimization problems; minimizing the energy thus minimizes the distortion. We also refer to meta-optimization techniques [Abulnaga et al. 2023; Poya et al. 2023], which describe symmetrization, respectively regularization methods for pre-existing distortion energies. Regardless of their efficiency (R2) and suitability for practical usage (R4), optimization-based methods generally lack *guarantees* of success in generating bijective maps (R1), unless they are already initialized with a bijective solution. *Preservation* of bijectivity can be ensured by a barrier term in the objective function, letting line search schemes detect invalid elements in the form of infinite energy. A typical use case is for deformation [Fang et al. 2021; Jiang et al. 2017], where a "rest" mesh is modified through small, incremental displacements. The incrementality here helps in preserving bijectivity throughout this sequence of target domains.

Conversely, some of these techniques, often referred to as *untangling* methods, can be initialized in a non-bijective way, and then attempt to obtain bijectivity through optimization. Notable and recent examples of such techniques, in which the target domain is prescribed in a per-vertex manner, include [Aigerman and Lipman 2013; Du et al. 2020; Garanzha et al. 2021; Su et al. 2019].

The importance of robust methods (R1) is verified by the ubiquity of the *Tutte embeddings* [Tutte 1963], which are guaranteed to produce valid 2D bijective maps for convex target domains. Those are generally poor in quality, yet their theoretical guarantees are leveraged by functioning as initializers for optimization-based methods, as mentioned above. Unfortunately, the Tutte embeddings lose their guarantees of bijectivity when applied to tetrahedral meshes [Alexa 2023].

2.2 Tetrahedral Maps with Guarantees

With Tutte's method not being applicable to the 3D realm, researchers have explored other robust approaches, some with promising results [Livesu 2020]. Robust methods are also being developed in 2D, e.g. with the *Advancing Front* method from Livesu [2024] or the *Progressive Embeddings* from Shen et al. [2019], for their strong guarantees, but also with the goal of extension to tetrahedral maps. However, tetrahedral meshes bring a geometric complexity that breaks critical assumptions of these methods. Specifically, in the context of Shen et al. [2019], the local mesh operation called "vertex split" is only guaranteed to be feasible under certain geometric conditions, which are (trivially) met in 2D but not in 3D. Nonetheless, Nigolian et al. [2023] succeeded in providing guarantees that hold in 3D. A key limitation of this work, as discussed in Section 1, is that the class of inputs it can handle is not well-understood. Indeed, one of their components performs a combinatorial exploration of subsets of vertices, which is deemed to have failed if no subset with specific properties can be found. Only in this case can a mesh be classified as an invalid input for this method. Unfortunately, even a relatively small number of vertices in the input mesh makes an exhaustive search of all subsets practically infeasible. It is therefore highly impractical to decide whether a specific input mesh is theoretically impossible to map with this method or if it would simply take an extended period of time.

Another approach with theoretical guarantees is [Campen et al. 2016], which generates continuous *foliation* maps. These are then converted into PL maps at the cost of sometimes extreme refinement. Hinderink and Campen [2023] extend foliation maps to star-shaped domains. Furthermore, for efficiency, they propose the local application of foliation maps to star-shaped subregions of the target domain (called *stars*), each encompassing flipped elements generated by an initial map (e.g. Tutte embedding). We note here that our method could be used as a drop-in replacement to individually handle these stars, considering that the setting fits the requirements of the SaE framework (ball-topology input, star-shaped prescribed boundary).

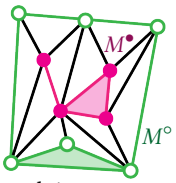
2.3 Refinement & Representation

One key observation regarding robust methods is that, contrary to their optimization-based counterparts, they generally heavily rely on punctual connectivity modifications. Indeed, most optimization-based methods do not include the connectivity of the input as a degree of freedom. Notable counterexamples include [Ferguson et al. 2023] in 3D and [Campen et al. 2021; Gillespie et al. 2021; Jin et al. 2014] in 2D, although these works differ from ours in their setting (in particular, non-prescribed boundary maps). Note that on-demand *remeshing* has also been leveraged for adaptive physical simulations [Anderson et al. 2005; Wicke et al. 2010]. Another advantage of robust methods is their compatibility with any numerical representation for their computations. While fixed-precision floating-point is the assumed standard for most applications, it is well-known that it can lead to issues, even for basic geometric predicates, e.g. triangle orientation. Seminal works, such as Shewchuk's predicates [Shewchuk 1997], show that great care has to be taken when dealing with such representation. In fact, even theoretically robust algorithms, such as the Tutte embeddings or Delaunay refinement, are known to be prone to failure when implemented with standard floating-point arithmetic [Finnendahl et al. 2023; Shen et al. 2019; Shewchuk 2002]. A common alternative is to represent (at least rational) numbers exactly, as *fractions*, with both numerator and denominator being integers of arbitrary size. A popular implementation for such rational-based representation is CGAL's `GMPQ` type [Fabri and Pion 2009], itself based on the GNU MultiPrecision (GMP) library. Relying on exact representations additionally resolves the sensitivity to almost degenerate states, e.g. those related to the geometric quality of the prescribed boundary conditions. The problem of recovering a floating-point-compatible result from the output of rational-based methods (e.g. to enable its use in floating-point-based downstream applications) is called *snap rounding* and is a notoriously difficult problem [Devillers et al. 2018], directly related to our precision requirement (R4).

3 CLUSTER MESHES

We first review some of the concepts used by SaE, before introducing original key concepts of our method.

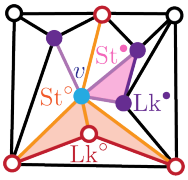
Expanded/Unexpanded Vertices: We split vertices into two sets: the *expanded* vertices, denoted as V°, and the *unexpanded* vertices, denoted as $V^\bullet$. Initially, the expanded vertices are the boundary vertices, and the unexpanded vertices are the (coincident) interior vertices. As is the case for SaE, our goal will be to *expand* vertices by moving them to a position inside the kernel of their 1-ring neighborhood. This requires their 1-ring neighborhood to have certain properties, as discussed later in this section. Once a vertex has been expanded, the $V^\bullet$ and V° sets are updated accordingly.



Cluster Mesh: We call the set of simplices consisting of only expanded vertices the *expanded mesh* and denote it as M°. Similarly, the set of simplices consisting of only unexpanded vertices is called the *unexpanded mesh*, denoted as $M^\bullet$. The figure on the right shows a 2D example, with the expanded mesh in green and the unexpanded mesh in magenta. Unexpanded vertices are shown as filled discs while expanded vertices are hollow to match the notation. Note that the unexpanded vertices geometrically coincide; we only explode them for visualization. Since all simplices of the unexpanded mesh form a cluster, we call it the *cluster mesh*. Note that $M^\circ \cup M^\bullet \neq M$, as some simplices have vertices in both sets. The cluster mesh will not, in general, be a *pure* simplicial complex. i.e. faces might be incident to no tetrahedron, and edges might be incident to no face. We do, however, require it to be simply connected. This, in addition to being the interior of a ball-topology mesh, implies its Euler characteristic to be that of a ball, i.e. 1. These properties bring a conceptual simplicity and are naturally maintained throughout the expansion process, as described in Sec. 3.2. Note that the cluster mesh consists of at least two vertices; expanding the second-last vertex automatically expands the last one. All incident simplices will already be positively oriented by construction. The cluster mesh represents the cornerstone of our novel approach, as we will only use its local connectivity information to check which vertices can be expanded.

Degenerate Simplices: An edge is geometrically degenerate if the positions of both vertices are identical. Similarly, a triangle is degenerate if all three vertices are collinear, and a tetrahedron is degenerate if all four vertices are coplanar. However, in the SaE framework, the only tolerated subset of geometric degeneracies are those caused by coincident (unexpanded) vertices, i.e. a degenerate simplex is always incident to an edge of zero length.

Stars, Closures & Links: Let us first recall standard mesh topology terms. First, the *star* of a vertex v, $\mathrm{St}(v)$, is the set of simplices incident to v in a given mesh. The *closure* of v, $\mathrm{Cl}(v)$ is the set of all vertices, edges, faces, and cells of $\mathrm{St}(v)$, including those not incident to v. Finally, the *link* of v, $\mathrm{Lk}(v)$, is the set of simplices of the closure of v that are not incident to v. Intuitively, $\mathrm{St}(v) \cap \mathrm{Lk}(v) = \emptyset$ and $\mathrm{Cl}(v) = \mathrm{St}(v) \cup \mathrm{Lk}(v)$. The figure below shows how those definitions can be extended to the expanded and cluster mesh. The *expanded star* $\mathrm{St}^\circ(v)$ is the set of non-degenerate simplices incident to v (which is their only unexpanded vertex).



The *unexpanded star* $\mathrm{St}^\bullet(v)$ is the set of simplices incident to v whose vertices are all coincident. Note that v itself is part of both $\mathrm{St}^\bullet(v)$ and $\mathrm{St}^\circ(v)$. On the left drawing, the coincident (unexpanded) vertices are drawn in purple, including v in blue. The degenerate simplices, i.e. the unexpanded star, are drawn in pink, while the expanded star is drawn in orange. Then, the expanded and unexpanded *links* of v, $\mathrm{Lk}^\circ(v)$ and $\mathrm{Lk}^\bullet(v)$, are drawn in red and purple, respectively, on the inline drawing. Figure 3 illustrates those concepts in 3D.

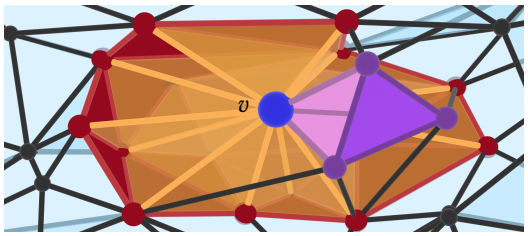


Fig. 3. A mesh cutout, with the large blue vertex v in focus. We see its *expanded star* $\mathrm{St}^\circ(v)$ in orange and *expanded link* $\mathrm{Lk}^\circ(v)$ in red, along with its *unexpanded star* $\mathrm{St}^\bullet(v)$ in pink and *unexpanded link* $\mathrm{Lk}^\bullet(v)$ in purple. The expanded star and link together form the *expansion cone* of v. Unexpanded stars and links are coincident in our method but are shown non-degenerate here for visualization only.

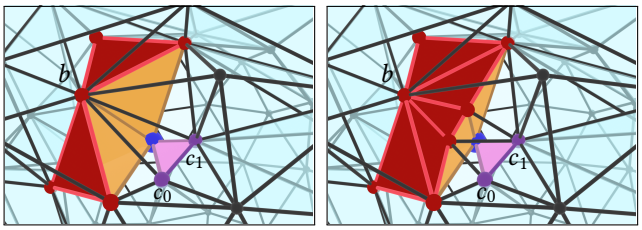


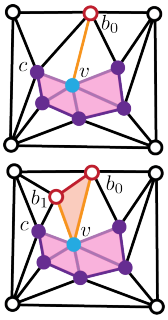
Fig. 4. A simple example of the *inflation* operation. Left: the blue vertex's expansion is non-manifold at vertex b, thus collapsing its kernel to a single edge. In [Nigolian et al. 2023], the blue vertex would not be considered to be topologically-expandable. Right: by splitting all edges (b, c_i), where c_i is any unexpanded vertex, the base of the expansion cone is now disc-topology and thus expandable. We hence consider the blue vertex as topologically-expandable, up to inflation.

3.1 Vertex Expandability

Let us now use those definitions to identify candidate vertices for expansion. In general, a vertex is expandable if it can be geometrically detached from the cluster mesh to a position that decreases the number of degenerate elements, without creating any inverted ones. This requires its expanded star to be *star-shaped*. As a reminder, we say that a domain is star-shaped if its kernel has non-zero volume. The kernel of a domain is the set of *guards* of the domain, points that are visible from any other point inside the domain. We first introduce a topological necessary condition for star-shapedness.

Topological Expandability: The expanded closure of a vertex v is called its *expansion cone* $\mathrm{EC}(v)$ in [Nigolian et al. 2023], and we will use the same terminology in this paper. In addition, this work defines a vertex to be *topologically-expandable* if its expansion cone is ball-topology and the base of its cone (the expanded link in our setting) is disc-topology. Expansion cones that are not 3-manifolds (with boundary) thus do not fit this expandability criterion. The reasoning is that the kernel of such cases would necessarily have zero volume, preventing the expansion of the corresponding vertex. Since we are only considering single vertices for expansion, and not subclusters, checking the cone base is sufficient; it being disc-topology necessarily implies that the cone is ball-topology. Moreover, we extend the definition of a *topologically-expandable* vertex to "a vertex whose expansion cone base is simply connected". Since a (non-pure) triangle mesh with boundary is simply connected if and only if it is a single connected component and Euler characteristic 1, we have that a vertex is topologically-expandable if and only if $\chi(\mathrm{Lk}^\circ(v)) = 1$ and $b_0(\mathrm{Lk}^\circ(v)) = 1$, where $b_0(S)$ is the 0th *Betti number*, i.e. the number of connected components.

Expansion Cone Inflation: Updating our definition of topological expandability to a larger class of expansion cones is made possible with a new local operation called *inflation*. If an expansion cone's base is not disc-topology, it necessarily means that one of its vertices is incident to multiple disconnected triangle fans, or none at all. An analogous situation in a 2D expansion cone (1D base) would be a vertex with a single edge, as shown in the inline figure below. Vertex v has a single vertex b_0 in its expansion cone's base and therefore has no space to expand. Again, the cluster mesh, coincident vertices are drawn in purple. The inflation is done by first splitting an edge between b_0 and cluster mesh vertex c. The newly created vertex b_1 can be moved so that the expansion cone of v is not degenerate anymore and v can be expanded. In 3D, this operation allows us to augment non-ball-topology expansion cones into ball-topology ones, as long as their base is simply connected (cf. the definition of topological expandability above). Figure 4 provides a more complete overview of this operation in 3D.



Simple Expandability: If a vertex is topologically-expandable and its expansion cone is star-shaped, we say it is *simply-expandable*. To bridge the gap between topological and simple expandability, we use the *star-shapification* operation from [Nigolian et al. 2023], Section 5.3. Conceptually, it relies on the fact that even if an expansion cone is not star-shaped, there must be a region lying inside of it that is. The initial non-star-shaped region can then be discretely bent to fit the star-shaped subregion, in a way that guarantees not to create any inversion. Discretizing this bending process is done by splitting edges connecting the vertex of interest and some of its expanded neighbors. Such edges can potentially be split numerous times, depending on the extent to which they must be bent to fit the star-shaped subregion. This potentially intense and very localized refinement will be one of the main source for slow run times, for reasons discussed in Sec. 4.2.2. Figure 5 provides an intuitive overview of this star-shapification operation. For a detailed description please refer to [Nigolian et al. 2023].

3.2 Expandability Criterion

So far, we have only expressed the expandability of a vertex in terms of its expansion cone. Let us now show how we can use information solely from the cluster mesh to obtain the same information.

Proposition 1. *The Euler characteristic of a (non-pure) mesh is constant under vertex removal if and only if the Euler characteristic of the link of the vertex is 1. i.e. if* $v \in M$, *then* $\chi(M \setminus St(v)) = \chi(M) \Leftrightarrow \chi(Lk(v)) = 1$.

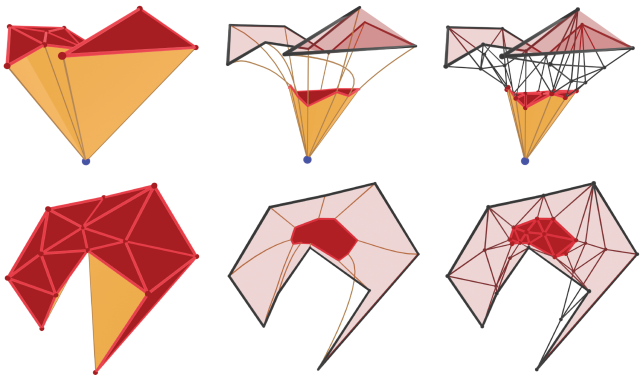


Fig. 5. An intuitive representation of the *star-shapification* operation presented in [Nigolian et al. 2023]. The top row shows the expansion cone of the blue vertex (here in isolation for clarity), while the bottom row depicts it seen from the top, highlighting the shape of its conic profile. Left: the expansion cone of the blue vertex is not star-shaped. Middle: by continuously bending some of the edges incident to the blue vertex, we can contract this shape into a star-shaped subregion of the original expansion cone. Right: this bending is discretized by splitting the corresponding edges, potentially multiple times. The relatively high number of additional vertices should hint at how much refinement such an operation can lead to, even for small expansion cones such as this one. By construction, the cells incident to the newly created vertices (not shown here for clarity) cannot be inverted.

Proof. Let v_v, e_v, f_v be the number of vertices, edges and faces of $\mathrm{Lk}(v)$. Then $\mathrm{St}(v)$ has 1 vertex, v_v edges, e_v faces and f_v cells. Now let V, E, F and C be the number of vertices, edges, faces and cells of M. When removing a vertex $v \in M$, we update M as $M \setminus \mathrm{St}(v)$. Its Euler characteristic is therefore:

$$\begin{aligned}\chi(M \setminus \mathrm{St}(v)) &= (V-1) - (E - v_v) + (F - e_v) - (C^\bullet - f_v) \\ &= (V - E + F - C) + (v_v - e_v + f_v) - 1 \\ &= \chi(M) + \chi(\mathrm{Lk}(v)) - 1 \\ &= \chi(M) \Leftrightarrow \chi(\mathrm{Lk}(v)) = 1\end{aligned}$$

□

In our case, we can replace M with $M^\bullet$, $\mathrm{Lk}(v)$ with $\mathrm{Lk}^\bullet(v)$ and "vertex removal" with "vertex expansion", as is the case for the following proposition:

Proposition 2. *A (non-pure) mesh remains a single connected component under vertex removal if and only if the link of the vertex is a single connected component. i.e. if $v \in M$, then $b_0(M \setminus St(v)) = 1 \Leftrightarrow b_0(Lk(v)) = 1$.*

Proof. For a given mesh with $b_0(M) = 1$, a vertex such that removing it from M would increase $b_0(M)$ is called a *cutvertex*. If v is not a cutvertex, then removing (expanding) it from M does not change $b_0(M) = 1$. If it is, then it necessarily is a neighbor of the multiple connected components of $M \setminus \mathrm{St}(v)$, which implies that $b_0(\mathrm{Lk}(v)) \neq 1$. Conversely, if $b_0(\mathrm{Lk}(v)) = 1$, then there must be a path between any two of its neighbors and v cannot be a cutvertex. □

Propositions 1 and 2 together imply that the cluster mesh is simply connected after expanding a vertex, if and only if the unexpanded link of this vertex is simply connected as well. We now move on to show the relation between the expanded link, i.e. the expansion cone base, and the unexpanded link.

Proposition 3. *If $v \in M^\bullet$, then $Lk^\circ(v)$ is simply connected $\Leftrightarrow$ $Lk^\bullet(v)$ is simply connected*

Proof. Since v is necessarily an interior vertex, its link must be of sphere-topology. The link is split into 3 parts: $\mathrm{Lk}^\circ(v)$, $\mathrm{Lk}^\bullet(v)$ and the interface between those two. Since $\mathrm{Lk}^\circ(v) \cap \mathrm{Lk}^\bullet(v) = \emptyset$, if either of those two has multiple connected components, then the other cannot be simply connected. Conversely, if either is not simply connected, the other must have multiple connected components. Therefore, one is simply connected if and only if the other is simply connected as well (see Figure 6). □

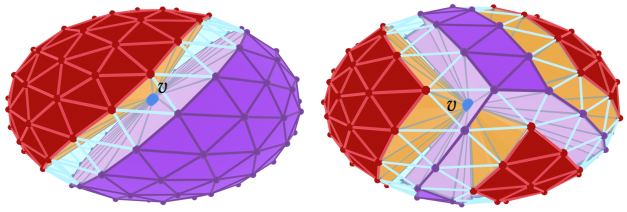


Fig. 6. Left: $\mathrm{Lk}^\bullet(v)$ (purple) is simply connected, and so is $\mathrm{Lk}^\circ(v)$ (red). The vertex in the center is therefore expandable. Right: $\mathrm{Lk}^\bullet(v)$ is not simply connected, and thus separates $\mathrm{Lk}^\circ(v)$ into multiple connected components. The vertex is therefore *not* expandable.

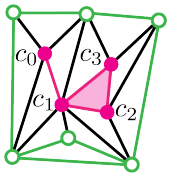
Corollary 1. *As long as we can find an unexpanded vertex whose unexpanded link is simply connected, we can expand it and maintain the simple connectedness of the cluster mesh.*

Proof. Trivial from Propositions 2 and 3. □

Thus, instead of using the simple connectedness of the expansion cone as in Sec. 3.1, we can use the following, algorithmically more efficient expandability criterion:

$$v \in M^\bullet \text{ is expandable} \Leftrightarrow b_0(\mathrm{Lk}^\bullet) = 1 \text{ and } \chi(\mathrm{Lk}^\bullet) = 1 \qquad (1)$$

We can relate this condition to *topological manifoldness* by observing that expandable vertices are always on the boundary of a topological k-manifold. A vertex is called *1-manifold-expandable* if it is incident to a single cluster mesh edge. When incident to a single open triangle fan of the cluster mesh, we call it *2-manifold-expandable*. Finally, a vertex incident to a single open half-ball of the cluster mesh is said to be *3-manifold-expandable*. As we can see on the right, with the cluster mesh in magenta and the expanded mesh in green, there is a direct relationship between the topological manifoldness of a vertex and its expandability. Vertices c_0, c_2, and c_3 are 1−, 2− and 2−manifold, have a simply connected unexpanded link (and cone base), and are thus expandable. However, c_1 is not manifold and has two connected components in its unexpanded link (and thus in its expansion cone base) and is hence not expandable. See Figure 7 for a more in-depth overview with a 3D example.



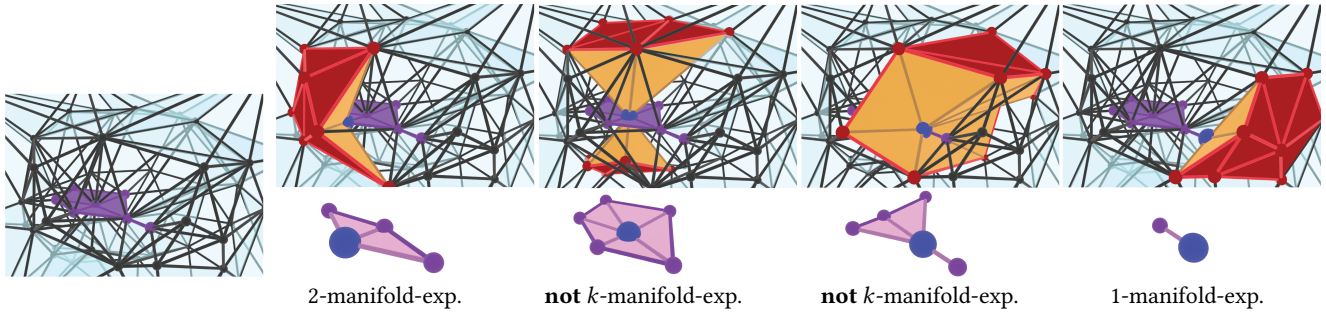


Fig. 7. Leftmost: a cutout of a mesh with its cluster mesh in purple (non-degenerate here for clarity). Then, from left to right, different vertices (in blue) and their expansion cones (in orange, base in red). Below each expansion cone example, the vertices' unexpanded stars and links. This illustrates the intrinsic relationship between the unexpanded link and the expansion cone base (the expanded link). The latter is simply connected if and only if the former is simply connected as well, as demonstrated in Sec. 3.2.

3.2.1 Main Advantage of the Cluster Mesh: The ability to focus on the cluster mesh highlights the main advantage of our approach. Indeed, although the expanded part of the mesh can be subject to arbitrary refinement, the cluster mesh is left unrefined throughout the expansion process. Since expansion cones can grow arbitrarily large due to refinement, using their properties as an expandability criterion means exploring an arbitrarily large number of vertices. On the other side, only requiring to explore the *unexpanded* vertices (from the cluster mesh) means that verifying the topological expandability criterion is bounded by the maximum valence of any vertex of the (unrefined) input mesh, warranting predictable and strictly bounded computation cost.

4 THE CLUSTER MESH EXPANSION ALGORITHM

In this section, we present our novel algorithm, which is a specialization of the SaE framework, shown in Alg. 1. Conceptually, our algorithm iteratively expands vertices from the cluster mesh until it becomes empty. In addition to selecting an expandable candidate vertex (line 5), each expansion might require inflation (line 9) and/or star-shapification (line 10) to enable a valid geometric position (line 11). Aside from the expansion order, two main factors influence the performance of our algorithm: the amount of refinement and the required numerical precision. In order to guarantee success, the inflation and star-shapification components require an exact representation of vertex coordinates. Since both components also refine the mesh, in the worst case, mesh complexity and memory requirements might increase exponentially. Consequently, the expansion candidate selection as well as all other components, are equipped with heuristics to minimize refinement and precision, and thus to maximize practical performance. In Sec. 4.1 we first clarify the expansion order of cluster mesh vertices, before detailing other important components to obtain good performance in Sec. 4.2.

4.1 Expansion Order

In each iteration of Alg. 1, our new criterion of Eqn. (1) characterizes the set of expandable vertices $Exp(M^\bullet)$ from the cluster mesh which could be processed next. Consequently, to warrant robustness, it is essential that $Exp(M^\bullet) \neq \emptyset$ whenever the cluster mesh $M^\bullet$ is non-empty. Otherwise, the algorithm could not proceed with expansions

Algorithm 1 The Cluster Mesh Expansion Algorithm

Input:
M, a ball-topology tetrahedral mesh
ϕ_∂, a bijective boundary piecewise linear map onto a star-shaped domain D
Output:
M', the input mesh M after potential refinement
ϕ, a bijective volumetric PL map of M' onto D

1: **procedure** ClusterMeshExpansion(M, ϕ_∂)
2: $\phi(V^\circ) \leftarrow \phi_\partial(V^\circ)$
3: $\phi(V^\bullet) \leftarrow$ Center(D)
4: **while** $V^\bullet \neq \emptyset$ **do**
5: $v \leftarrow$ SelectCandidate($M^\bullet$) //Sec. 4.1.2
6: **if** no candidate found **then**
7: NOBUENOSS detected → STOP
8: **end if**
9: $(\phi, M') \leftarrow$ Inflate($EC(v)$) //Sec. 3.1
10: $(\phi, M') \leftarrow$ Star-Shapify($EC(v)$) //Sec. 3.1
11: $\phi(v) \leftarrow$ Center($EC(v)$) //Sec. 4.2.3
12: $M^\bullet \leftarrow M^\bullet \setminus \mathrm{St}^\bullet(v)$
13: $V^\bullet \leftarrow V^\bullet \setminus v$
14: **end while**
15: **end procedure**

despite there still being degenerate elements. Interestingly, when experimenting with random expansion orders on our entire dataset of 12000 inputs, we did not find any failure case, suggesting that at least the probability of generating a non-empty cluster mesh with $Exp(M^\bullet) = \emptyset$ is very low. Intuitively, this can be understood by recalling from Sec. 3.2 that expandable vertices sit on the boundary of a k-manifold, and the existence of a simply connected, non-pure simplicial complex without k-manifold-expandable vertices is not obvious. Please also note the direct relation to the class of shellable meshes, where a shelling sequence warrants the iterative removal of elements while preserving simple connectedness. Hence, it is not surprising that from a given shelling sequence, a viable expansion sequence can always be constructed, as we explain in Sec. 4.3. However, it is somewhat surprising that non-empty cluster meshes with $Exp(M^\bullet) = \emptyset$ do exist, as described next.

4.1.1 NOBUENOSS. We clarify the existence of a non-pure simplicial complex that is a "**NO**-**BO**Undary, **E**uler-characteristic-1, **N**on-manif**O**ld, **S**imply-connected **S**urface" by explicit construction. It cannot consist solely of vertices and edges, since its simple connectedness induces a tree structure, which always exhibits 1-manifold-expandable vertices at its leaves. However, as depicted in Fig. 8, starting from a disc-topology surface and then gluing the surface along its boundary, the simple connectedness requirement can be preserved, while turning the boundary into a non-manifold curve. Interestingly, this particular example is topologically equivalent to the (subdivided) interior of the *Knotted Hole*, a famous non-shellable mesh introduced one century ago by Furch [1924].

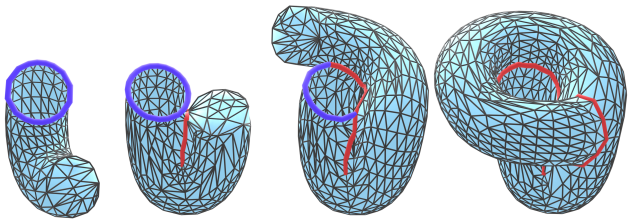
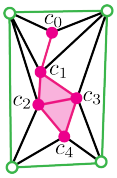


Fig. 8. An example of a "**NO**-**BO**Undary, **E**uler-characteristic-1, **N**on-manif**O**ld, **S**imply-connected **S**urface" (NOBUENOSS) and how it can be constructed. The boundary edges (blue) are turned into non-manifold edges (red) by gluing them to another part of the mesh. In the resulting mesh, none of the vertices is k-manifold-expandable. The resulting surface is 3D-embeddable, simply connected and does not separate space but still does not have k-manifold boundary points. The model is available at https://www.algohex.eu/publications/cluster-mesh-sae

4.1.2 Expansion Priority. While, according to our experiments, the selection of a specific expansion candidate from $Exp(M^\bullet)$ is not critical for robustness, it might still significantly impact the overall run time due to induced mesh refinement and precision requirements. Consequently, we devise a heuristic selection mechanism with the goal of minimizing refinement and precision demands, consisting of three criteria. The number one priority is simply-expandable vertices, as they do not require any refinement during expansion. If there is no simply-expandable vertex, we next consider vertices requiring star-shapification, and then those requiring inflation, both types involving on-the-fly refinement. To further disambiguate candidates, a second selection criterion is employed, preferring k-manifold-expandable vertices with lower k. This can be seen as trying to keep the cluster mesh as close to a pure simplicial mesh as possible. Indeed, removing a 1-manifold-expandable vertex



(e.g. c_0 on the left) can only improve the situation for its sole neighbor, potentially making it expandable, as is the case for c_1 here. Expanding 2- and 3-manifold-expandable vertices, however, can negatively impact the expandability of their neighbors. An example is given by c_2 and c_3 on the left, where both are initially expandable, but expanding either leaves the other non-expandable. The third and final selection heuristic to further disambiguate is the number of expanded neighbors of a candidate vertex. We observed that prioritizing simply-expandable vertices with a higher number of vertices in their expansion cone is beneficial in the long run. Such vertices might otherwise require star-shapification in the future, with an amount of refinement proportional to the number of neighbors. Conversely, the expansion cone of a vertex requiring star-shapification, should be as small as possible, to minimize the necessary refinement.

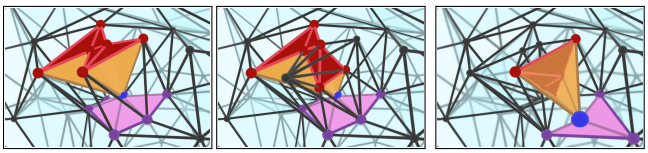
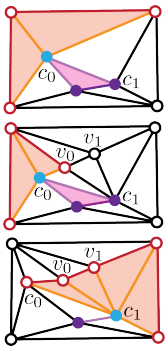


Fig. 9. Left: a vertex (blue) and its non-star-shaped expansion cone. Middle: the robust star-shapification operation (cf. Sec. 3.1 and Fig. 5) refines some edges on its sides. Right: the additional vertices created by those edge splits are now part of another expansion cone. The potential star-shapification of this second cone will thus require even more splits, which will in turn impact the expansion of the subsequent vertices. In addition, using exact representation means that newly created vertices require an exponentially increasing number of bits for their representation.

4.2 Implementation Details

4.2.1 Expansion Cone Inflation. The inflation operation, introduced in Sec. 3.1 and illustrated in Figure 4, is implemented as follows. Consider a given vertex v whose expansion cone is not of disc-topology (but is simply connected), thus requiring inflation: For each vertex $b \in \mathrm{Lk}^\circ(v)$ that is a cutvertex of $\mathrm{Lk}^\circ(v)$, split all edges (b, c_i) such that $c_i \in M^\bullet, c_i \neq v$. Those topological splits initially create sets of coincident vertices, which are subsequently expanded by re-location inside the kernel of their 1-ring neighborhood. This geometric positioning part of the inflation is identical to the "Cluster Isolation" operation from [Nigolian et al. 2023], Appendix C.1, which is guaranteed to expand all split vertices sequentially.

4.2.2 Post Star-Shapification Edge Collapses. The star-shapification procedure is conservative in generating enough degrees of freedom, often splitting a single edge multiple times. Consequently, we inherit the edge collapsing mechanism of [Nigolian et al. 2023] to greedily remove additional vertices that turn out to be unnecessary. As seen in the figure to the right, splitting an edge on the boundary of the expansion cone of vertex c_0 creates vertices v_0 and v_1, which will become part of the expansion cone of c_1. In a 3D case, this would increase the likelihood of c_1's expansion cone to require star-shapification *and* the cost to do so. See also Figure 9 for a 3D example. Therefore, the edge-collapsing operation, even though relatively time-consuming, can sometimes efficiently counterbalance the refinement necessary to perform future star-shapifications.



4.2.3 Precision Reduction. In our algorithm, a given vertex v, with $\mathrm{EC}(v)$ being star-shaped, is expanded to its *Chebyshev center*, which is a center offering a kernel-inscribed ball of maximum radius. Note

that the Chebyshev center can be computed through a linear program and the explicit construction of the kernel is never required in our algorithm. However, the exact position of the Chebyshev center is often of unnecessarily high precision since any (low-precision) point from the kernel would be sufficient. Consequently, we inherit the precision truncation mechanism of [Nigolian et al. 2023], which, based on sampling, searches low-precision points near the Chebyshev center and within the kernel. We refer the reader to [Nigolian et al. 2023] for all implementation details of this component, which we do not modify in any way.

Therefore, we add a novel precision reduction mechanism, specifically targeting expansions involving star-shapification. The star-shapification first contracts the initial expansion cone towards an edge connecting v and a so-called *witness vertex* w, an expanded neighbor of v. Consequently, there must be a point somewhere between v and w that is in the kernel of $\mathrm{EC}(v)$. Sampling low-precision convex combinations on this line segment empirically performs better than naive sampling near the Chebyshev center. We employ a simple binary-search strategy, starting from the midpoint between v and w. The search terminates if a valid position is found or if the required precision becomes excessive, i.e. higher than the precision of the Chebyshev center.

4.2.4 Smoothing. Similarly to [Nigolian et al. 2023] we employ Laplacian smoothing to foster the creation of star-shaped expansion cones and prevent vertices from geometric clustering inducing high precision requirements. However, since this operation can be costly when applied globally, it is preferable to perform smoothing more selectively. Empirically, only smoothing the cluster mesh's direct (expanded) neighbors provides a good trade-off between run time cost versus precision reduction, and the number of expansion cones made star-shaped.

4.2.5 Subcluster Expansions. Although expanding single vertices is theoretically sufficient for our novel algorithm, our experiments indicated that expanding multiple vertices at once can sometimes be beneficial. Indeed, the bottleneck of our method is the star-shapification operation; it should be avoided as much as possible. When considering multiple candidate vertices simultaneously, we can investigate the union of their expansion cones and verify star-shapedness. In such a case, those vertices can be expanded to a location in this union's kernel and be subsequently expanded individually. Note that we do not extend our novel definition of topological expandability (see Sec. 3.1) to subclusters and instead only rely on the geometric criterion of star-shapedness. Picking a valid subset of unexpanded vertices is a difficult combinatorial problem, and we thus restrict our subsets to consist of at most 4 vertices. We justify this choice of an upper bound with the experiment shown in Figure 16. Generally, this process, named a *non-trivial cluster expansion* and described in detail in [Nigolian et al. 2023] implies systematically less refinement than the star-shapification. Even if enabling this component results in a clear increase in performance, it comes with the unfortunate additional time complexity of its combinatorial nature. We discuss the benefits of its integration in more detail in Sec. 5.4. An additional important note here is that although we do not provide any proof, it is clear that enabling the expansion of subclusters would not help in making progress in a NOBUENOSS configuration. Indeed, in such a case, the expansion cone of any (strict) subset of vertices would also have multiple connected components on its boundary, making this subset not expandable.

4.2.6 The Priority Queue Cluster Mesh Expansion Algorithm. Let us now discuss how those components can be assembled into an efficient implementation. First, we set up a priority queue, such that vertices are handled in an order according to the criteria presented in Sec. 4.1.2. The queue is initially filled with all k-manifold-expandable cluster mesh vertices (and their corresponding priority). In addition, whenever a vertex is expanded, we also reevaluate the priority of its direct neighbors and update the priority queue accordingly. This allows to keep a polynomial average-case asymptotical behavior for the candidate exploration component. However, accounting for the punctual star-shapification operations, the worst-case asymptotical behavior can become exponential. We thus see star-shapifications as a last resort and only apply them after expiring all simply-expandable vertices *and* 2-, 3- and 4-subclusters expansions. Sec. 5.4 discusses how these two distinct behaviors manifest in our test cases. Finally, the smoothing pass described in Sec. 4.2.4 is only done periodically, after each 10 expansions, offering a good compromise between the number of vertices made simply-expandable and the smoothing overhead.

4.2.7 Bijectivity In IEEE 754 Double Precision. While using an exact data type to represent the coordinates of vertices offers strict guarantees, e.g. to find valid centers during expansion, it is often not suitable for real-world applications. Specifically, relying on exact rational numbers prevents us from using some mathematical functions (e.g. the square root), which might be necessary for downstream applications. In order to bridge the gap between bijectivity in exact and floating-point representation, we apply a final pass of dedicated geometric optimization on the resulting mesh. It consists in iteratively moving each vertex to the Chebyshev center of its 1-ring neighborhood. A single pass on all interior vertices corresponds to a Gauss-Seidel iteration of the optimization problem of maximizing the minimum tetrahedral heights. An alternative approach would be to optimize the tetrahedral heights directly, but using an iterative Gauss-Seidel approach allows us to benefit from the precision reduction mechanism of Sec. 4.2.3. In our experiments, a final pass of snap rounding is sufficient to obtain bijective maps in double precision for the majority of inputs.

4.3 Expansion Sequences for Shellable Meshes

As mentioned in Sec. 4.1, encountering a NOBUENOSS cluster mesh, and thus not being able to expand all vertices, is highly unlikely. Nevertheless, we are able to devise a method to entirely avoid reducing the cluster mesh into such a configuration, as long as the input mesh is *shellable*. As a reminder, a ball-topology mesh with n cells is shellable if there exists a sequence of cells $c_0, c_1, ..., c_n$ such that the mesh remains ball-topology after each removal of a cell in sequence. From such a sequence, we can generate an expansion sequence as follows: for each cell c_i of a shelling sequence, we pick one vertex of c_i such that expanding it removes *exactly* c_i from the cluster mesh. This can only happen if the c_i is incident to a single interior face. Otherwise, we split either an edge or a face of c_i, then expand this

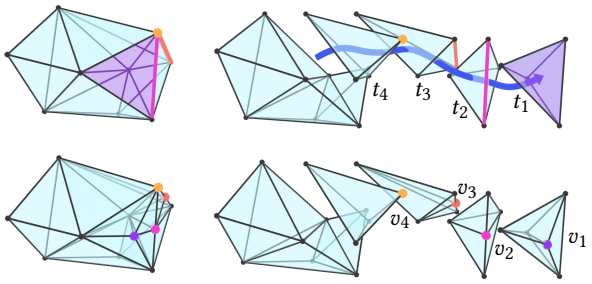


Fig. 10. Top right: Part of a shelling sequence (t_1, t_2, t_3, t_4) of the top left mesh. This shelling sequence can be converted to the bottom right expansion sequence (v_1, v_2, v_3, v_4) by first splitting faces or edges based on the number of interior faces of the corresponding cells. The first removed cell t_1 has 3 interior faces, and its only boundary face (purple) is split. By removing this new vertex v_1, the resulting mesh is the same as if removing the original cell. Subsequently, splitting the edge incident to the two boundary faces of t_2 (magenta) generates vertex v_2, which can be deleted to mimic the deletion of t_2. If a cell only has a single interior face, such as t_4, removing the opposite vertex v_4 effectively deletes t_4 without requiring any refinement.

additional vertex, effectively removing the (now subdivided) cell c_i from the cluster mesh. This new vertex is guaranteed to be expandable by construction. An important subtlety here is that to obtain this expansion sequence from the shelling sequence, we temporarily consider boundary vertices as unexpanded, meaning the whole mesh is considered to be the cluster mesh. "Expanding" boundary vertices is simply done by assigning them their prescribed position. Figure 10 illustrates the conversion from shelling to expansion sequence, and Algorithm 2 formally defines this process. However, the downside of pre-computing the expansion sequence is that it does not consider the "priority" (as described in Sec. 4.1.2). As a worst case, the sequence could entirely comprise vertices requiring star-shapification, leading to an exponential asymptotical behavior.

Algorithm 2 A Shelling-Based Expansion Sequence

Input:
M, a shellable, ball-topology tetrahedral mesh
Output:
S_{exp}, a sequence of expandable vertices
M', the input mesh M, refined so that S_{exp} is a viable expansion sequence

```
procedure ShellingBasedExpansionSequence(M)
    S_exp ← ()
    S_shell ← FindShellingSequence(M)
    M_shell ← M
    for each t ∈ S_shell do
        k ←#interior faces of t inside M_shell
        Let v be the next vertex to expand
        if k = 1 then
            v ← the vertex opposite to the interior face of t in M_shell.
        else if k = 2 then
            Split the only edge not incident to either of the two interior
            faces of t in M_shell.
            v ← the vertex created by this edge split.
        else if k = 3 then
            Split the only boundary face of t in M_shell.
            v ← the vertex created by this face split.
        end if
        Append v to S_exp
        M_shell ← M_shell \ t
    end for
end procedure
```

5 EVALUATION

In this section, we evaluate various aspects of our algorithm and provide a detailed comparison to the state of the art. We specifically compare our novel algorithm, referred to as Cluster Mesh (CM), to publicly available implementations of the original Shrink-and-Expand (SaE) method[1] [Nigolian et al. 2023] and the Galaxy Maps (GM) method[2] [Hinderink and Campen 2023]. Note that we omit a direct comparison to other relevant approaches, such as TLC [Du et al. 2020] or the Foldover-Free maps [Garanzha et al. 2021], since the data is already available in [Nigolian et al. 2023]. However, it should be noted that while many mapping techniques, including TLC and FoF, are rather sensitive to the distortion of the prescribed boundary, for our method we could not observe a systematic run time improvement when reducing the boundary distortion in a pre-processing step.

Dataset. Unless explicitly stated differently for all experiments we employ the dataset of Nigolian et al. [2023], which consists of ~3k ball-topology tetrahedral meshes originating from the TetWild dataset [Hu et al. 2018]. Each of those is equipped with 4 distinct and challenging star-shaped boundary conditions, resulting in a total of ~12k test cases.

Time Limit. In order to handle the large dataset, the run time of each input is restricted to 6 hours. We observed that most inputs that do not finish within 6 hours, still do not terminate after several days. Responsible for such worst-case behavior is repeated refinement during star-shapification, e.g. depicted in Figure 9, and precision blow-up, which both can lead to exponential run time complexity.

5.1 Expansion Sequence Viability

The first experiment verifies that our novel vertex expansion criterion of Eqn. 1 is suitable for greedily finding viable expansion sequences. For this experiment, we only consider single-vertex expansions (no clusters) and drop all geometric requirements, including star-shapification and inflation. This way, the worst-case time complexity resulting from refinement and precision blow-up can be effectively avoided, and the entire dataset can be successfully processed within 120 minutes. Consequently, the greedy expansion order of Alg. 1 is well-justified, not reducing the overall success rate but effectively avoiding the additional costs of the theoretically superior shelling-based alternative of Sec. 4.3. This experiment suggests that all timeouts, when considering the geometric requirements, result from refinement and precision blow-up. Moreover, NOBUENOSS surfaces were never found, except for the manually created Knotted-Hole case of Fig. 8. A theoretically fully satisfactory

[1] https://github.com/cgg-bern/expansion-cones
[2] https://github.com/SteffenHinderink/GalaxyMaps

Table 1. The table below summarizes key metrics for our comparison with the original SaE implementation. Perhaps the most important one is the success rate, showing the percentage of meshes for which a bijective map could be obtained within the 6h time limit, in exact representation. Our algorithm here performs significantly better, even though the time limit is still exceeded for 4.2% of cases. Another important metric for practical uses is the rate at which maps can be converted to *floating-point* (f.p.), making our new version of SaE more compatible with downstream applications. Comparing run times, averaged per cell of an input mesh and then per mesh, our method is about twice faster than the original SaE. It also requires significantly less refinement to obtain valid maps. The speed and refinement comparison is further developed in Figure 11.

	success < 6h	success f.p.	time / input tet (s) avg	time / input tet (s) max	growth ratio avg	growth ratio max
SaE	76.3%	36.6%	$2.31\ 10^{-2}$	2.56	1.66	459
CM (ours)	**95.8%**	74.3%	$6.36\ 10^{-3}$	1.18	1.06	34.2

algorithm can be obtained by executing the shelling-based alternative of Sec.4.3 whenever Alg. 1 terminates with a NOBUENOSS. Moreover, even the shellability limitation can be resolved by allowing on-demand global refinement whenever no shelling sequence exists [Campen et al. 2016]. However, while theoretically appealing, the practical relevance of such an algorithm is unfortunately rather limited due to its excessive worst-case run time.

5.2 Comparison with original Shrink-and-Expand

Since the input requirements of SaE and our novel CM are identical, the comparison is straightforward. We compare w.r.t. (R1) the practical robustness as the number of bijective maps constructed within 6 hours, (R4) the precision requirements as the ability to truncate the result to IEEE 754 double precision, (R2) the run time as the average time per input tetrahedron, and (R3) the parsimony as the *growth ratio*, i.e. the ratio between the number of vertices of the output (after refinement) and input mesh. Table 1 and Figure 11 clearly demonstrate that CM outperforms SaE w.r.t. all these key metrics, boosting the success rate within a 6h time limit from 76.3% to 95.8%. Furthermore, as shown in Table 2, there are only a few cases of relative underperformance, where SaE delivered a bijective map within the time limit while CM could not. Another key result is demonstrated in Figure 12, which suggests that the average asymptotical run time behavior of CM is polynomial rather than the observed exponential behavior of SaE. Regarding the distortion requirement (R5), the results of CM are similar to those of SaE. In Sec. 5.4.2 we show that extreme distortion sometimes limits the value of the generated maps as initializers for subsequent distortion minimization. All bijective maps generated with CM are available at https://www.algohex.eu/publications/cluster-mesh-sae, for further analysis, visualization or comparison.

5.3 Comparison with Galaxy Maps

Galaxy Maps (GM) [Hinderink and Campen 2023] consist of two major stages. Starting from a given map with degeneracies, the first stage isolates degeneracies by covering them with star-shaped cavities in the image. The second stage then employs a variant of the

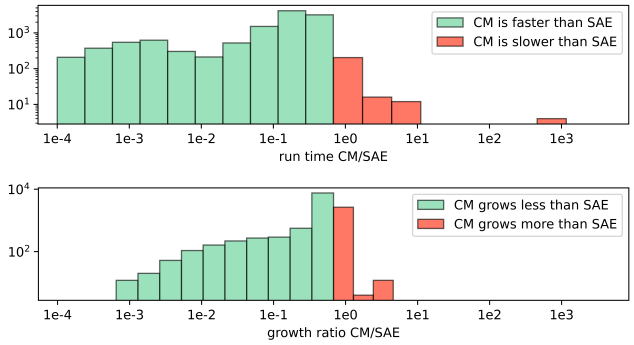


Fig. 11. We measure our performance compared to the original SaE implementation by computing the per-mesh ratio for the run time and growth ratio. CM performs significantly better in both metrics, being faster than SaE in 98% of cases and growing less in 86.6%. We note, however, that in some relatively rare cases, our implementation can be significantly slower than the original SaE.

Table 2. Looking at the outcome for each mesh of the dataset between the original SaE implementation and ours, we see that our implementation has a significantly higher success rate. Moreover, among the cases for which our method times out before finding a bijective map within 6 hours, the reference method can only successfully map 12 of them. We could not identify any particular feature in these 12 cases. Interestingly, there are 490 meshes for which both methods time out, indicating that those are the most challenging cases of the dataset.

		CM (ours) timeouts	CM (ours) successes	total
SaE	timeouts	490	2334	2824
	successes	12	9061	9073
total		502	11 395	11 897

Foliation (FOL) mapping approach [Campen et al. 2016] to obtain a bijective map for each star-shaped cavity, effectively resolving all defects of the given initial map. Since the isolation of defects is also possible in combination with our CM approach, we keep the first stage but replace FOL with CM in the second stage. As hundreds of stars of various sizes can occur in a single mesh, we restrict the comparison to the 60 cuboids taken from [Brückler et al. 2022b] and used in the evaluation of GM. Extracting the stars generated by GM from those 60 meshes, we obtain 1784 test cases, with sizes ranging from 12 to 4081 tetrahedra (~47.9 on average). We generate PL maps for each of those using FOL and compare the results to those of CM, as summarized in Table 3 and Fig. 13. The success rates are similar, but CM clearly outperforms FOL w.r.t. run time and refinement. Consequently, replacing the naive shrinkage-based initialization of CM with the more sophisticated growing of stars of GM offers a powerful combination to efficiently and robustly generate bijective tetrahedral maps. Interestingly, when running GM on our dataset, for most challenging cases where CM times out, GM determines a single large star, i.e. it is unable to isolate defects of smaller size. Hence, those elusive 490 hard cases (see Table 2) form an interesting dataset to stress-test future methods.

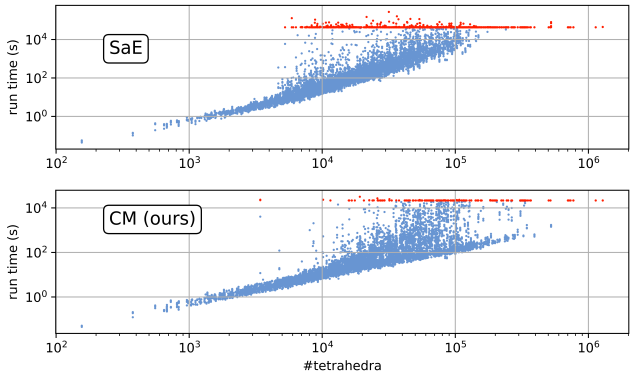


Fig. 12. Each dot of this scatter plot represents a single mesh. Blue dots are cases where a bijective map was obtained within the 6h time limit, while the red dots exceeded the time limit. Aside from the lower number of red dots, this clearly suggests that our method shows polynomial dependence on the input size on average rather than exponential dependence as the original SaE implementation. However, a significant portion of test cases still show exponential behavior, as we discuss further in Sec. 5.4.

Table 3. A summary of the comparison with the foliation method [Campen et al. 2016]. Although we have pretty similar success rates, both in exact and floating-point (f.p.) representations, the performance gain is clear. Our method is about 30 times faster and requires significantly less refinement.

	success < 6h	success f.p.	time / input tet (s) avg	time / input tet (s) max	growth ratio avg	growth ratio max
Fol	99.4%	99.3%	$1.56\ 10^{-2}$	6.82	2.11	214
CM (ours)	**99.5%**	99.3%	$4.90\ 10^{-4}$	0.01	1.004	2.51

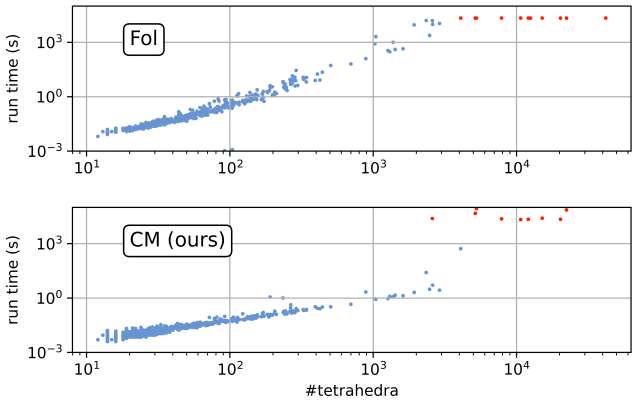


Fig. 13. Looking at the relation between mesh size and performance, it is apparent that our method performs better than the foliation method. Although FOL is more consistent in its trend, it becomes highly impractical for meshes (or star cavities) larger than a few thousand tetrahedra.

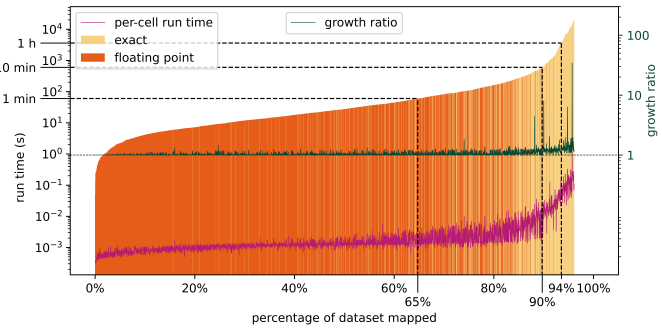


Fig. 14. Sorting each mesh by its run time, we can see what percentage of the dataset can be mapped within some given duration. Dark orange bars indicate meshes for which a floating-point bijective map was extracted from the exact representation one, while light orange bars indicate cases for which the latter was necessary. Notably, one minute is sufficient to generate a bijective map for most meshes. Past the inflection point around the one-minute mark, meshes become increasingly costly to map. Even though ~90% of cases can be mapped within 10 minutes, handling the rest becomes exponentially costly. We also note how both the per-cell average run time and growth ratio follow a trend similar to the per-mesh run time.

5.4 Additional Experiments

5.4.1 Detailed Analysis of CM. The cumulative plot of Figure 14 shows that a large fraction of inputs can be processed significantly faster than the time limit of 6h, 65% within 1 minute, and 90% within 1 hour. Figure 15 provides additional insights regarding the most time-consuming parts for success versus timeout cases. For inputs requiring a large number of star-shapifications the frequently unsuccessful subcluster search often becomes a critical bottleneck. However, as discussed in Sec. 4.2.5, expanding subsets of vertices (subclusters) can also sometimes significantly improve the performance, as illustrated in Fig. 16. In summary, according to our experiments, it seems clear that the principal source of refinement and precision blow-up is the star-shapification component inherited from SaE.

5.4.2 Distortion Minimization Initialization. According to Sec. 2.1, the bijective maps generated by CM are valuable as initialization of optimization-based techniques. We illustrate this principle by optimizing a symmetric Dirichlet barrier energy (see [Abulnaga et al. 2023]) to reduce the distortion of bijective maps generated by CM. Of the 2846 bijective maps to a prescribed tetrahedral boundary (a subset of our dataset) that we obtain within the time limit, the distortion can be successfully reduced for 1430 cases. The conformal and volumetric distortion before and after the optimization are illustrated in Figure 17. All remaining cases contain at least one highly distorted element, which is geometrically valid when investigated with exact predicates but numerically evaluated to infinite energy since the Jacobian determinant in IEEE 754 double precision is zero or negative. Consequently, from the perspective of practical applications, better bounding the distortion of the CM approach will be an important direction for future work.

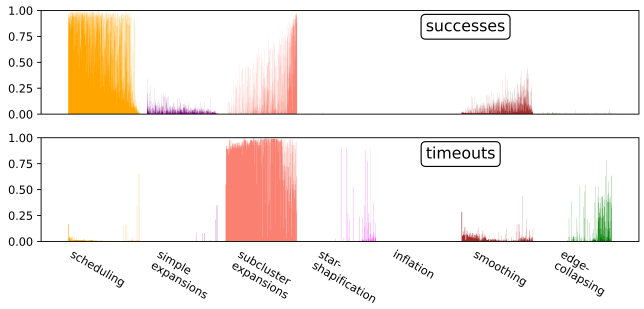


Fig. 15. An overview of the time distribution between the different parts of the algorithm. Each mesh of the dataset is represented by a set of bars, sorted by their total run time. Splitting this data between success and timeout cases highlights the difference in behavior. For success cases, there are two main dominant categories. Close to the entirety of the time is sometimes spent on scheduling (maintaining the expansion priority queue, computing the expandability of candidate vertices, etc.), which typically happens for fast-processed meshes. For longer run times, on the other side, the most dominant part is condidate subclusters exploration. However, in case of a timeout, our method spends significant effort on exploring subclusters, often without finding a valid one. Additionally, star-shapifications can sometimes be quite costly, most likely when dealing with large and/or high-precision expansion cones. The precision reduction mechanisms described in Sec. 4.2.3 can also represent a significant portion of the total computations, but are necessary to maintain reasonable overall run times.

5.4.3 Map Generalization and Hexahedral Meshing. We demonstrate the practical value of CM for generating bijective maps subject to general non-ball-topology domains and w.r.t. non-star-shaped boundary constraints. For this experiment, we rely on the motorcycle complex based algorithm of [Brückler and Campen 2023; Brückler et al. 2022a,b], constructing a bijective integer-grid map for hexahedral mesh generation. The algorithm starts from a given bijective seamless map, then extracts the motorcycle complex and performs an integer-quantization including collapses in the cell-complex to eventually obtain conforming boundary constraints for a set of otherwise independent cuboid maps. CM offers a robust solution for constructing these maps, where in each case, a deformed cuboid needs to be mapped bijectively to an integer-sized and axis-aligned cuboid in the image. Figure 18 illustrates the approach, where individual cuboids are shown in different colors.

5.5 Future Work

The refinement and precision blow-up, mainly originating from the star-shapification operation, sometimes lead to the non-satisfactory worst-case run time behavior of the CM algorithm. Consequently, it would be essential for future star-shapification operations to better exploit the available geometric degrees of freedom by combining a deformation targeting star-shapedness with very selective splits, instead of the conservative splitting scheme employed so far. Another limitation of Alg. 1, which deserves future investigations, is the inability to directly and efficiently handle NOBUENOSS cluster meshes without the theoretical workaround of Sec. 5.1. While those seem to be extremely rare in real-world meshes, the topological restriction might still be problematic for certain applications.

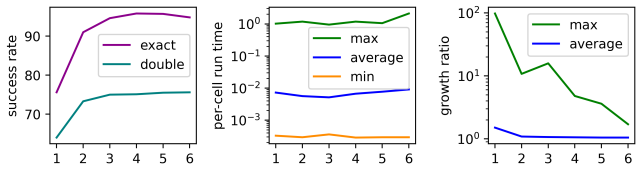


Fig. 16. Evolution of three key performance metrics, depending on the maximum number of vertices that can be expanded at once ("subcluster", see Sec. 4.2.5). Using larger subclusters is clearly advantageous in terms of success rate (both in exact and floating-point representation) and refinement, while not meaningfully impacting run time. We do note, however, that using subclusters of size larger than 4 actually has a negative impact on performance, while not improving success rate. This can simply be explained by the fact that exploring k-subclusters is a component with $O(n^k)$ complexity, n being the number of vertices of the cluster mesh. Another important note regards the decreasing maximum growth rate (green curve, right plot). This significant decrease comes from the worst-growth meshes not being mapped within the time limit anymore. Using larger subclusters thus prevents reaching the point where star-shapifications (and their corresponding intense refinement) are necessary.

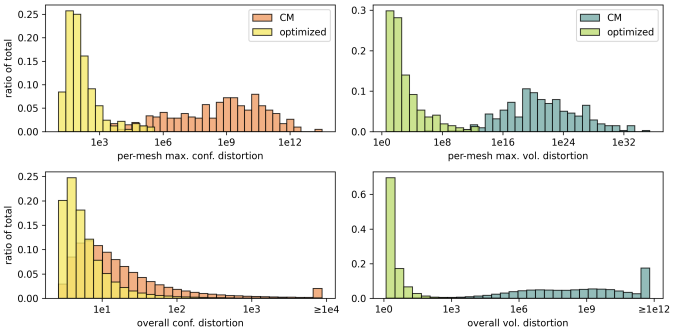


Fig. 17. We measure the per-element distortion in two ways: the *conformal* distortion $\mathrm{tr}(J^TJ)/\det J$, and the *volumetric* distortion $\det J + 1/\det J$, with J being the map's per-element Jacobian matrix. This histogram shows the distribution of conformal (left) and volumetric (right) distortion, in terms of the ratio of the total number of meshes (top) and total number of elements (bottom) both before and after optimizing our maps with a symmetric Dirichlet energy term. This data comes from 1430 cases for which our method generates maps that can be used as an nitializer for further optimization.

6 CONCLUSION

In this work, we presented our Cluster Mesh Expansion algorithm to generate bijective maps from ball-topology meshes to star-shaped prescribed boundaries. As a specialization of a recent bijective tetrahedral mapping framework, it extends the theoretical guarantees on the class of input meshes it can successfully map. Evaluated on a large and challenging dataset, it also proved to be more efficient than the state of the art in terms of robustness (R1), run time (R2), parsimony (R3), and precision (R4), as defined in Sec. 1. Despite a small subset of remaining hard cases, where the practical run time is prohibitively high, the CM algorithm is already suitable for many practical applications.

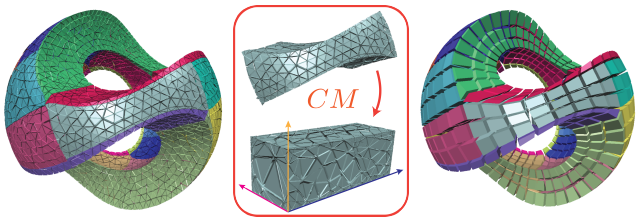


Fig. 18. Through the motorcycle complex of [Brückler et al. 2022b], a genus 3 mesh (left) is decomposed into ball-topology blocks, each mapped using our method (CM) to an integer-grid-aligned parameter space (middle). Due to the conforming quantization of [Brückler et al. 2022a] the integer-grid map induced hexahedra of individual blocks stitch to a conforming hexahedral mesh (right).

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). On the part of M. Campen, this work was funded by the Deutsche Forschungsgemeinschaft (DFG) - 497335132. The authors would also like to thank H. Brückler and S. Hinderink for their help regarding the experiment depicted in Fig. 18 and the Galaxy Maps comparison, respectively.

REFERENCES

S Mazdak Abulnaga, Oded Stein, Polina Golland, and Justin Solomon. 2023. Symmetric volume maps: Order-invariant volumetric mesh correspondence with free boundary. *ACM Trans. Graph.* 42, 3 (2023), 1–20.

Noam Aigerman and Yaron Lipman. 2013. Injective and Bounded Distortion Mappings in 3D. *ACM Trans. Graph.* 32, 4 (2013).

Marc Alexa. 2023. Tutte embeddings of tetrahedral meshes. *Discrete & Computational Geometry* (2023), 1–11.

Anthony Anderson, Xiaoming Zheng, and Vittorio Cristini. 2005. Adaptive unstructured volume remeshing – I: The method. *J. Comput. Phys.* 208, 2 (2005), 616–625.

Hendrik Brückler and Marcel Campen. 2023. Collapsing Embedded Cell Complexes for Safer Hexahedral Meshing. *ACM Trans. Graph.* 42, 6 (2023).

Hendrik Brückler, David Bommes, and Marcel Campen. 2022a. Volume parametrization quantization for hexahedral meshing. *ACM Trans. Graph.* 41, 4 (2022).

Hendrik Brückler, Ojaswi Gupta, Manish Mandad, and Marcel Campen. 2022b. The 3D motorcycle complex for structured volume decomposition. In *Computer Graphics Forum*, Vol. 41. 221–235.

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

Marcel Campen, Cláudio T. Silva, and Denis Zorin. 2016. Bijective Maps from Simplicial Foliations. *ACM Trans. Graph.* 35, 4 (2016).

Gianmarco Cherchi and Marco Livesu. 2023. VOLMAP: a Large Scale Benchmark for Volume Mappings to Simple Base Domains. In *Computer Graphics Forum*, Vol. 42. e14915.

Olivier Devillers, Sylvain Lazard, and William Lenhart. 2018. *3D snap rounding*. Ph. D. Dissertation. Inria Nancy-Grand Est.

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* (Seattle, Washington) *(GIS '09)*. Association for Computing Machinery, New York, NY, USA, 538–539.

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M Kaufman. 2021. Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.* 40, 4 (2021).

Zachary Ferguson, Teseo Schneider, Danny Kaufman, and Daniele Panozzo. 2023. In-Timestep Remeshing for Contacting Elastodynamics. *ACM Trans. Graph.* 42, 4 (2023), 1–15.

Ugo Finnendahl, Dimitrios Bogiokas, Pablo Robles Cervantes, and Marc Alexa. 2023. Efficient Embeddings in Exact Arithmetic. *ACM Trans. Graph.* 42, 4 (2023), 1–17.

Robert Furch. 1924. Zur grundlegung der kombinatorischen topologie. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, Vol. 3. Springer, 69–88.

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), 1–16.

Mark Gillespie, Boris Springborn, and Keenan Crane. 2021. Discrete conformal equivalence of polyhedral surfaces. *ACM Trans. Graph.* 40, 4 (2021), 1–20.

Steffen Hinderink and Marcel Campen. 2023. Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Trans. Graph.* 42, 4 (2023), 1–16.

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), 60.

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph.* 36, 6 (2017).

Yao Jin, Jin Huang, and Ruofeng Tong. 2014. Remeshing-assisted optimization for locally injective mappings. In *Computer Graphics Forum*, Vol. 33. 269–279.

T. Kanai, H. Suzuki, and F. Kimura. 1997. 3D geometric metamorphosis based on harmonic map. In *The Fifth Pacific Conference on Computer Graphics and Applications*. 97–104.

Charalampos Koniaris, Darren Cosker, Xiaosong Yang, and Kenny Mitchell. 2014. Survey of texture mapping techniques for representing and rendering volumetric mesostructure. *Journal of Computer Graphics Techniques* (2014).

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

Yaron Lipman and Thomas Funkhouser. 2009. Möbius Voting for Surface Correspondence. *ACM Trans. Graph.* 28, 3 (2009).

Heng Liu and David Bommes. 2023. Locally Meshable Frame Fields. *ACM Trans. Graph.* 42, 4, Article 112 (jul 2023), 20 pages. https://doi.org/10.1145/3592457

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.). The Eurographics Association.

Marco Livesu. 2024. Advancing Front Surface Mapping. *Computer Graphics Forum* 43 (2024), e15026.

Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. Cubecover–parameterization of 3d volumes. In *Computer graphics forum*, Vol. 30. 1397–1406.

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

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

Roman Poya, Rogelio Ortigosa, and Theodore Kim. 2023. Geometric optimisation via spectral shifting. *ACM Trans. Graph.* 42, 3 (2023), 1–15.

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).

Hanxiao Shen, Zhongshi Jiang, Denis Zorin, and Daniele Panozzo. 2019. Progressive embedding. *ACM Trans. Graph.* 38, 4 (2019).

Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18 (1997), 305–363.

Jonathan Richard Shewchuk. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry* 22, 1 (2002), 21–74. 16th ACM Symposium on Computational Geometry.

Jian-Ping Su, Xiao-Ming Fu, and Ligang Liu. 2019. Practical foldover-free volumetric mapping construction. In *Computer Graphics Forum*, Vol. 38. 287–297.

William Thomas Tutte. 1963. How to draw a graph. *Proceedings of the London Mathematical Society* 3, 1 (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.

Jiaran Zhou, Changhe Tu, Denis Zorin, and Marcel Campen. 2020. Combinatorial construction of seamless parameter domains. In *Computer graphics forum*, Vol. 39. 179–190.