

Robust Deformation-based 3D Bijective Mapping through Differential Boundary Shape Matching

VALENTIN Z. NIGOLIAN, University of Bern, Switzerland

MARCEL CAMPEN, University of Paderborn, Germany

DAVID BOMMES, University of Bern, Switzerland

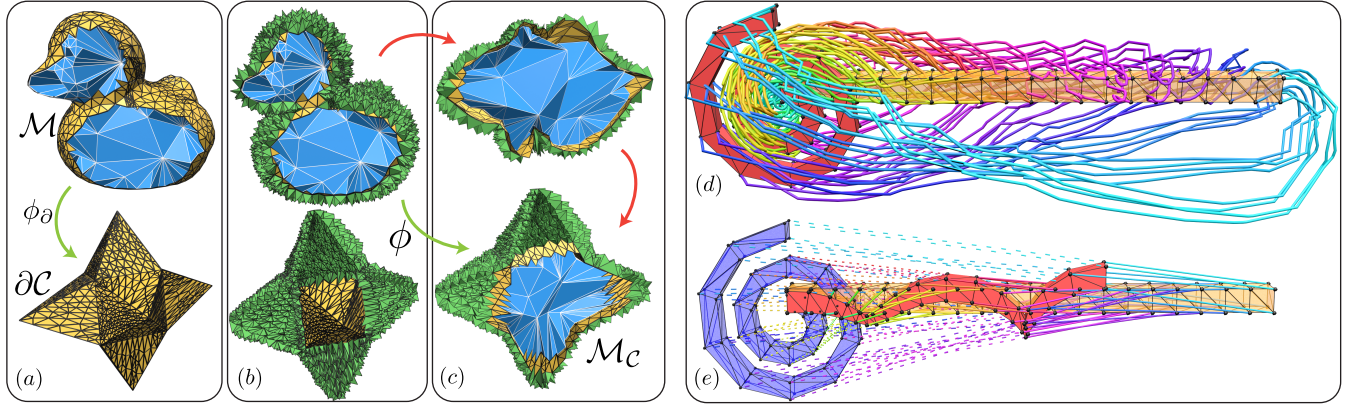


Fig. 1. (a) Given a solid shape (with tetrahedralisation $\mathcal{M}$) and a boundary map ϕ_{∂} onto a target space boundary $\partial\mathcal{C}$, our method completes this to a volumetric map ϕ , in a piecewise linear manner. It is based on performing a deformation towards the target shape. (b) As its key to success, this deformation is not driven by explicit positional objectives; instead our approach compatibly augments the mesh, in its input shape and in its target shape, with a layer of boundary-incident virtual tetrahedral elements (green). (c) Continuously deforming each virtual element into its target shape while dragging $\mathcal{M}$ along, yields a global deformation of $\mathcal{M}$ into the target shape mesh $\mathcal{M}_C$. (d, e) For a different instance (rod to spiral): trajectories of boundary vertices under the deformation successfully driven by our technique (d) or by a classical positional objective (e) where near-linear movement leads into a blocking state (red).

We introduce a novel framework to generate piecewise linear tetrahedral maps, by expressing the map-finding problem as a constrained mesh deformation problem, where constraints satisfaction implies both injectivity and target shape conformity. Deformation is supported by the two core components of our approach: First, an augmentation of the mesh by boundary-incident virtual elements, encoding the target shape in a differential manner via its first and second fundamental form; in contrast to position-driven approaches this prevents optimisation failure due to improper winding. Second, a remeshing scheme maintaining mesh element quality and preventing blocking configurations. A first implementation is empirically shown to have higher robustness and/or higher efficiency than state-of-the-art methods on challenging, yet practically relevant inputs. We illustrate our method's versatility in a range of applications, demonstrating its value as a new item in the algorithmic toolbox for volumetric mapping.

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

Additional Key Words and Phrases: tetrahedral mesh, volumetric maps, bijection, homeomorphism, mesh deformation, remeshing

Authors' Contact Information: Valentin Z. Nigolian, valentin.nigolian@gmail.com, University of Bern, Bern, Switzerland; Marcel Campen, marcel.campen@upb.de, University of Paderborn, Paderborn, Germany; David Bommes, david.bommes@unibe.ch, University of Bern, Bern, Switzerland.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 1557-7368/2026/12-ART206
<https://doi.org/10.1145/3842573>

ACM Reference Format:

Valentin Z. Nigolian, Marcel Campen, and David Bommes. 2026. Robust Deformation-based 3D Bijective Mapping through Differential Boundary Shape Matching. *ACM Trans. Graph.* 45, 6, Article 206 (December 2026), 15 pages. <https://doi.org/10.1145/3842573>

1 Introduction

Generating dense correspondences between two 3D objects, so-called *maps*, is an essential aspect of many digital geometry applications, such as volumetric texturing, shape analysis, physics-based deformation or hexahedral meshing. A necessary property for most applications is that of *bijectivity* (or *local injectivity* for self-overlapping cases), in particular the absence of degeneracies or even inversions. Of particular interest are *piecewise linear* maps, represented on a tetrahedral mesh. Essentially, a tetrahedral mesh in two different poses, with two sets of vertex coordinates, covering two spatial objects defines the map; if all elements are positively oriented, the map is free of degeneracies and inversions.

In such a mapping problem, the input is a piecewise linear source (domain) region $\mathcal{D} \subset \mathbb{R}^3$, with a prescribed boundary map onto the boundary of a target (codomain) region $\mathcal{C} \subset \mathbb{R}^3$. The sought volumetric map is a function $\phi : \mathcal{D} \rightarrow \mathcal{C}$, represented on a tetrahedral mesh $\mathcal{M}$ of domain $\mathcal{D}$, obtained by computing a second embedding $\mathcal{M}_C$ (i.e. a second set of vertex coordinates for $\mathcal{M}$) such that its boundary $\partial\mathcal{M}_C$ conforms to the codomain boundary $\partial\mathcal{C}$. Importantly, $\mathcal{M}$ and $\mathcal{M}_C$ differ only in their embedding; their connectivity is the same, thereby immediately defining a piecewise linear map

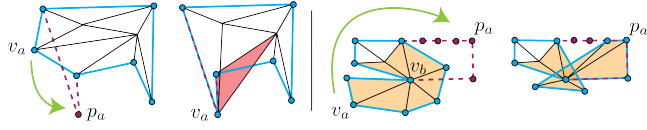


Fig. 2. The two ways the boundary deformation can get stuck (in 2D for clarity). Boundary edges are blue, dashed purple lines indicate the target shape ∂C . Left: by moving towards its target position p_a , vertex v_a would necessarily flip the orientation of a triangle. Right: with a clockwise deformation path for vertex v_a , it would meet its boundary position p_a , but create a double-cover around center vertex v_b by its incident faces (orange), with inner angles summing to more than 2π . The remaining two vertices now can no longer reach their target.

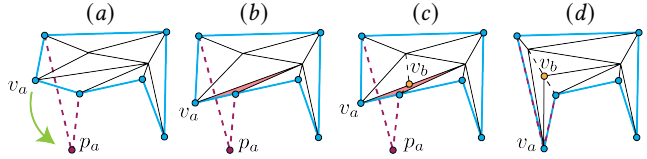


Fig. 3. (a): Vertex v_a cannot move to its target position p_a without flipping at least one element (cf. Figure 2, left). (b): Enforcing flip prevention still allows near-degenerate elements to form. (c): We use the high-distortion of this element as an indicator that connectivity modifications must be made, here in the form of an *edge split*, producing a new vertex v_b . (d): This additional degree of freedom lets v_a move towards its target position.

between $\mathcal{D}$ and $\mathcal{C}$. The map is considered *valid* if all tetrahedra are positively oriented, both in domain and codomain (V1), and we have $\partial M_C = \partial C$ (V2).

Recent volumetric mapping methods conceptually fix the boundary of sought M_C to the codomain shape, trivially meeting condition (V2), and spending the algorithmic effort on computing interior vertex positions that meet (V1). In this work, we take the opposite approach, considering mapping as a deformation problem, where we initially set $M_C = M$, thus trivially meeting condition (V1), and continuously morph M_C to ultimately meet condition (V2).

Such a deformation approach comes with one major challenge: constraints that are set up to preserve positive orientation (V1) throughout, may prevent reaching a state satisfying (V2). This actually comes in two flavours:

- There are insufficient degrees of freedom, i.e. the discretisation is (fundamentally or transiently) inadequate. See Figure 2 left.
- There is a local minimum at the boundary of the feasible set, i.e. the satisfaction of (V2) cannot further be monotonically tightened, at least not without (temporarily) giving up constraints (V1). See Figure 2 right.

We tackle these two issues as follows.

Remeshing: To address the first issue, we note that in such a blocking situation the deformation will push some tetrahedra into a near-degenerate (zero volume) state, presenting high distortion. We use this information as a criterion to decide where to make local connectivity changes in the interior, as illustrated by Figure 3, adapting

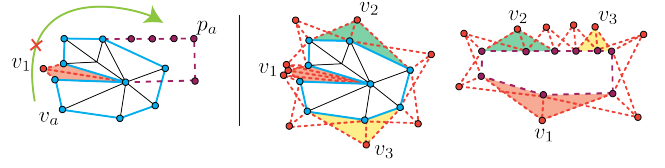


Fig. 4. Left: by adding a virtual vertex v_1 , and virtual triangles (red) connecting it to boundary edges, we close the triangle fan incident to vertex v_b . This prevents vertex v_a from moving to its target position p_a clockwise, as it would flip the orientation of an element. In 3D, the virtual elements would be tetrahedra closing the fan of a boundary *edge*. Right: We can add such virtual elements for each boundary vertex (boundary edge in 3D), both in domain and codomain. Three virtual vertices and their incident virtual elements are highlighted as examples. Deforming every virtual element into its target (codomain) shape, the positional boundary conditions given as input would be met (up to rigid transformation).

the initial discretisation M as needed. Where distinction is necessary, we will refer to the remeshed version as M' . This remeshing process focuses on locally reconfiguring tetrahedra to decrease their distortion, creating more space for vertices to move. Importantly, remeshing is performed such that both embeddings of the mesh, M' and the incrementally deforming M_C' , remain valid at all times, and the boundary remains unaffected, $\partial M' = \partial M = \partial \mathcal{D}$.

In scenarios where the mesh M is not just constructed ad hoc to represent the map, but there is an intentional predefined mesh of domain $\mathcal{D}$, such general alterations of the interior mesh structure (even if possibly fundamentally inevitable) can be undesirable. In Appendix A we remark on how one can effectively restrict this to conforming refinement only.

Fundamental Form Guidance: To tackle the second issue, we encode the target shape (and the objective to reach this shape) not in a direct positional manner, but in a differential manner, via its boundary's first and second fundamental forms, capturing intrinsic metric and extrinsic curvature. In the discrete setting, these can be understood as edge lengths and dihedral angles, and are known to completely describe a shape [Wang et al. 2012]—up to rotation and translation. It is this invariance that is the key to success here.

Instead of directly targeting edge lengths and dihedral angles in the deformation process, we augment the mesh M with a layer of (overlapping, only locally conforming) virtual tetrahedral elements, as illustrated in Figure 4. These are designed such that when they reach their individually assigned target shape in the deformation process, this implies the discrete fundamental forms have globally reached the desired state, i.e. condition (V2) is met (up to a global rigid transformation). Importantly, this even implies that the dihedral angles are matched properly, not just modulo 2π as in some other discretisations. Furthermore, this augmentation allows us to formulate the deformation constraint conveniently as a simple distortion minimisation objective over these virtual tetrahedra.

The virtual elements are constructed, locally for each boundary edge, such that they essentially close the edge's fan of incident tetrahedra to a full wheel (Figure 4). It is this loop closure that enables us to, merely by maintaining these elements' positive orientation

just like we do for the actual tetrahedra, prevent improper winding as pictured in Figure 2 from emerging at any point during the deformation.

1.1 Contributions

At its core, our approach consists in solving a sequence of constrained optimisation problems focused on the virtual elements' conformity to their target shape until all boundary vertices have reached their target position, interleaved with remeshing passes. Its conceptual simplicity creates a modularity that opens the door to a variety of implementations, of which we provide a concrete example, described in Section 4. This algorithm is evaluated in Section 5.

Although its runtime efficiency does not reach that of some alternative methods, it shows a favourably high degree of practical robustness to difficult inputs. It appears to be particularly well-suited for numerically or conceptually challenging cases, where other methods either fail, time out, or require extended numerical precision.

The code of our implementation and the employed datasets are available at <https://www.algoheX.eu/publications/deformation-maps>.

2 Related Work

2.1 Piecewise Linear Tetrahedral Maps

Despite considerable effort in recent years, tetrahedral mapping remains a difficult problem for which no single method offers all practically desired properties (in particular efficiency and reliability) in all circumstances and applications. In most cases, the first step of modern methods is to conceptually set the boundary vertex positions to their intended target position on $\partial\mathcal{C}$, leaving the effort to finding a valid embedding for the interior of $\mathcal{M}$. There are two broad method categories to deal with this: *untangling* methods, which attempt to recover positive element orientation through a numerical optimisation formulation, and *constructive* methods, explicitly computing vertex image positions in a way that guarantees bijectivity.

While methods of the first group, such as [Du et al. 2020; Garanzha et al. 2021; Overby et al. 2021] are convenient for their conceptual simplicity, performance, and ability to generate low-distortion maps, they do not offer guarantees regarding map bijectivity. On the other hand, methods of the second group, such as [Campen et al. 2016; Hinderink and Campen 2023; Nigolian et al. 2023] do offer theoretical guarantees, but suffer from practical limitations, most notably in terms of induced mesh refinement, with consequences also for worst-case runtime. Aside from this, they also tend to generate very *distorted* maps, where tetrahedra can have extremely different shapes between source and target spaces, with negative impact on downstream applications' performance. Moreover, these are often limited to specific input classes, such as star-shaped codomains or ball-topology meshes, unless combined with extension techniques [Hinderink et al. 2024; Hinderink and Campen 2026], while optimisation-based methods typically have less restrictions.

The theoretical guarantees of the aforementioned constructive methods partially hinge on the ability to modify the connectivity of their input meshes, through a sequence of local mesh operations

that fall under the umbrella term of *remeshing*. Indeed, it is easy to construct cases where a fixed mesh is missing the degrees of freedom required to conform to a given target shape, as illustrated in Figure 3 left.

The complementarity characteristics of these two categories of approaches is the reason for constructive methods often being considered *initialisers*, providing a bijective initial map which can then be used as input by optimisation-based methods that reduce distortion while preserving positive orientation [Rabinovich et al. 2017]. For a broader overview of maps and their intricacies, we refer the reader to [Naitsat et al. 2021] and [Meloni et al. 2024].

2.2 Deformation & Remeshing

The idea of phrasing mapping problems as deformation problems is not new, with techniques like the one introduced by [Fang et al. 2021]. This work also takes an optimisation-based approach, with one of the energy terms being the positional constraints violation $\|X - X^*\|^2$, with X being the vertices' current positions, and X^* their prescribed target positions. The overall process is split into a sequence of deformation steps, with intermediate target positions $X^0, X^1, \dots, X^*$. If generated through linear interpolation, one of the strategies used in their paper, it can lead to the issue illustrated in Figure 1(e) and further discussed in Section 5.7. As an alternative, they propose the use of non-linear paths for the boundary vertices; this mode, however, relies on user interaction.

Related to our mesh augmentation, a kind of virtual tetrahedral elements have been employed to express deformation gradients [Sumner and Popović 2004]. Differently from our technique, though, these are individual tetrahedra per boundary face, do not fully encode dihedral angles, and are used to approximately transfer an existing deformation rather than to drive one.

Other deformation-related methods maintain numerical stability of elasto-plastic deformation simulations through systematic remeshing [Ferguson et al. 2023; Wen et al. 2025; Wicke et al. 2010]. This stems from the ubiquitous observation that simulations involving the Finite Element Method (FEM) suffer from a loss of accuracy when element distortion becomes high. How distortion is defined concretely is application-dependent, suggesting that methods should be modular to accommodate for different use cases. Generally, though, a tetrahedron's dihedral angles are an important quality indicator, and directly relate to discretisation error and numerical stability [Shewchuk 2002]. Distortion metrics hence usually include these angles, explicitly or implicitly, for instance through conformal distortion or element aspect ratio [Sorgente et al. 2023].

Regardless of the quality metric choice, deciding when, where, and how to remesh is a challenge, as discussed in the literature on the topic, both in 2D [Dunyach et al. 2013; Jin et al. 2014] and 3D [Klingner and Shewchuk 2007; Plaza and Rivara 2003]. Even more so in the mapping setting, where all connectivity changes must maintain positive orientation throughout both the domain and codomain meshes. We will discuss our own choices in Section 4.5.3, and for now just note that we take a conservative approach akin to [Klingner and Shewchuk 2007], applying operations globally in a greedy manner, with performance being left to secondary consideration.

In principle, remeshing could also be applied in optimisation-based mapping methods employing untangling [Du et al. 2020; Garanzha et al. 2021; Overby et al. 2021], so as to generally facilitate success even for initially inadequate meshes. However, it remains open how to meaningfully measure quality on partially degenerate or inverted meshes so as to systematically decide where what kind of remeshing is necessary or beneficial. This supports our approach of maintaining positive element orientation throughout the deformation process.

3 Boundary-Deformation-Driven Mapping Framework

3.1 Problem Formulation

To formalise our optimisation problem, let us now define $X \in \mathbb{R}^{3(n+m)}$, our search space consisting of the 3D positions for the n vertices of $\mathcal{M}_C$, and the m additional virtual vertices, as described in Figure 4. We then define a function $d : \mathbb{R}^{3(n+m)} \rightarrow \mathbb{R}^+$ encoding the distortion of the virtual elements w.r.t. to their target shape, and a function $f : \mathbb{R}^{3(n+m)} \rightarrow \mathbb{R}^+$ encoding the interior distortion of $\mathcal{M}_C$, either w.r.t. to $\mathcal{M}$, or some other reference shape. Together, they form the following optimisation problem:

$$\begin{aligned} \min_X \quad & f(X) \\ \text{s.t.} \quad & d(X) = 0 \\ & \min \text{vol}(X) > 0, \end{aligned} \quad (1)$$

where $\text{vol}(X)$ denotes the per-element signed volumes. Finding a feasible solution of this optimisation problem necessarily yields a map satisfying (V1) and (V2).

3.2 Numerical Stability

So far, our discussion omitted the preoccupations inherent to floating-point representation, a cornerstone of most geometry processing applications. Indeed, it is well-known that simple theoretical notions, like shape intersections or element orientation, can be inaccurately computed when using f.p. [Shewchuk 1996]. This extends to numerical optimisation, which very often involves computing the first- and second-order derivatives of objective and constraint functions, themselves being rather sensitive to ill-conditioning. Just a few extremely badly-shaped elements can drive these quantities to ranges that are such that the optimisation computations become meaningless, thus blocking deformation progress. For this reason, we will integrate a set of *stability guards* to our framework, to maintain numerical soundness throughout the optimisation process. Namely, we add a quality bound, an edge-length bound, and an element volume bound, integrated into our optimisation problem (1) as follows:

$$\begin{aligned} \min_X \quad & f(X) \\ \text{s.t.} \quad & d(X) = 0 \\ & \min \text{vol}(X) > v_{\min} \\ & \min \text{qual}(X) > q_{\min} \\ & \min \text{len}(X) > l_{\min}, \end{aligned} \quad (2)$$

with $\text{qual}(X)$ and $\text{len}(X)$ being the per-element qualities, and per-edge lengths, respectively. $v_{\min}$, $q_{\min}$ and $l_{\min}$ are thus parameters of our framework; note that it is always possible to choose these such that they are satisfied initially.

3.3 Penalty Formulation

Among the many ways to solve such a feasibility problem, we transform our problem (2) into a *penalty* formulation. Constraints are shifted to the objective function, with log-barrier terms used to maintain the stability guards, as follows:

$$\begin{aligned} \min_X \quad & \nu f(X) + \mu d(X) \\ & - \lambda_v \sum_{\tau \in T} \log(\text{vol}(\tau) - v_{\min}) \\ & - \lambda_q \sum_{\tau \in T} \log(\text{qual}(\tau) - q_{\min}) \\ & - \lambda_l \sum_{e \in E} \log(\text{len}(e) - l_{\min}), \end{aligned} \quad (3)$$

with $\nu, \mu, \lambda_v, \lambda_l, \lambda_q \in \mathbb{R}^+$, T the set of tetrahedra of $\mathcal{M}$, and E the set of edges of $\mathcal{M}$. Assuming the guards are initially satisfied (by picking the bounds according to the input), the corresponding weights can be set to very low values, thus having minuscule effect unless $\min \text{vol}(X)$, $\min \text{qual}(X)$ or $\min \text{len}(X)$ approach their respective bounds. We pick $\lambda_v, \lambda_l, \lambda_q = 10^{-3}$.

At its core, our method consists of solving problem (3) using a Newton-based line-search method, then applying a remeshing strategy to unlock the deformation, adapting ν and μ , and repeating until $d(X)$ becomes sufficiently small. Importantly, our line-search prevents tunnelling through inverted tetrahedra, by finding the roots of a degree-3 polynomial, similarly to [Smith and Schaefer 2015]. The complete scheduling additionally involves performance-oriented features, and is discussed in detail in Section 4.

4 Differential Advancing Front Mapping (DAF)

In this section, we first give an overview of the various components and technical considerations of our implementation (Algorithm 1), which is then given explicitly at the end of the section.

4.1 Virtual Elements Setup

Virtual tetrahedral elements are created individually for each boundary edge of $\mathcal{M}$, so as to complete its fan of incident tetrahedra to a full wheel. Crucially, it is not necessary to coordinate between adjacent boundary edges to build a *conforming* layer of virtual elements, nor to prevent overlaps between these elements and other parts of the mesh. This would actually imply intricate challenges not addressable with existing conforming scaffolding techniques [Jiang et al. 2017] or based on controlled offsetting [Zint et al. 2025] because in our setting the mesh needs to be covered in both embeddings (domain and codomain) *compatibly*, i.e. the virtual tetrahedra need to have positive orientation in both. The individual construction per boundary edge, of non-conforming overlapping elements suffices and significantly simplifies the matter.

Completing the open tetrahedral fan of a boundary edge can be done with a variable number of virtual tetrahedra, as shown in Figure 5, with the lower limit dictated by the smaller of the initial and the target dihedral angle of that edge. As two virtual elements (with one virtual vertex) are suitable for the angle range $(0, 2\pi)$, we generally use this configuration, avoiding further case distinction for simplicity. The virtual vertex position is determined by minimizing the virtual elements' conformal distortion relative to

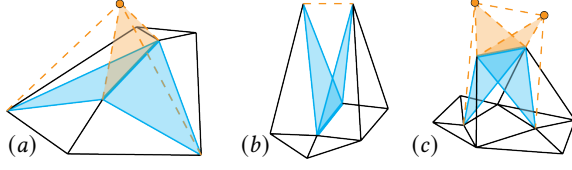


Fig. 5. (a) The (inner) dihedral angle at the boundary edge of interest (blue, thick) is below π , so two virtual elements (with one virtual vertex) are necessary to close the gap. (b) The dihedral angle is above π , so a single virtual elements would be sufficient. (c) The dihedral angle is close to 0; two elements would be sufficient, but more than two would be better shaped.

a regular tetrahedron. As shown on the right, for a boundary edge (a, b) , incident to two boundary faces (a, b, c) and (b, a, d) , the virtual vertex is initialised at v_0 by rotating point d by an angle of δ , half-way towards plane (a, b, c) around axis (a, b) , yielding positively orientated virtual elements. This computation, `INITIALISEVIRTUALVERTICES()` in Algorithm 1, is done at the very beginning of the method for all boundary edges (∂E) , once based on the domain boundary vertex positions (to augment $\mathcal{M}$) and once based on the codomain boundary vertex positions (to determine the target shape for these virtual elements).

4.2 Quality & Distortion Functions

As distortion metric $\text{qual}(X)$ for the stability guards, we pick the *volume-length* [Parthasarathy et al. 1994], defined as

$$\text{qual}(\tau) = 6\sqrt{2}V / l_{\text{RMS}}^3,$$

where V is the volume of tetrahedron τ , and l_{RMS} is the root-mean-square of its 6 edge lengths. As interior regulariser $f(X)$ we use the same metric,

$$f(X) = \sum_{\tau \in \mathcal{T}} \text{qual}(\tau)^{-1},$$

which will help in keeping the mesh away from the quality bound $q_{\min}$. For the function $d(X)$, supposed to measure the difference of virtual elements from their target shape, we use the Symmetric Dirichlet energy,

$$d(X) = \sum_{\tau \in \mathcal{T}} (||J_\tau||_F^2 + ||J_\tau||_F^{-2} - 6),$$

where J_τ is the linear map Jacobian of virtual tetrahedron τ from the set of virtual tetrahedra $\hat{\mathcal{T}}$. This choice is motivated by the fact that it encodes both conformal (dihedral angles) and volumetric (edge lengths) information, and is minimal only if all virtual elements are congruent to their target state.

4.3 Guard Parameters

Values chosen for $v_{\min}$, $q_{\min}$ and $l_{\min}$ should be rather small, so as to impede deformation as little as possible, yet large enough to prevent numerical instability in the optimiser. Empirically, we choose $v_{\min}$ as 10^{-6} times the average codomain element volume, $l_{\min}$ as 10^{-3} times the minimum codomain boundary edge length, $q_{\min}$ (due to being scale invariant) constantly at 10^{-6} . If some element of an input mesh exceeds any of these bounds, they are raised locally for this element, so as to yield an initially satisfying state.

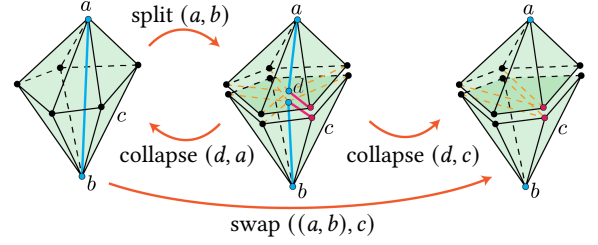


Fig. 6. Local edge remeshing operators. By *splitting* the edge (a, b) on the left, we create a new vertex d , as shown in the middle (with partially-expanded cells for visualisation). *Collapsing* the edge (d, a) or (d, b) recovers the initial situation. Alternatively, chaining the split (a, b) with the collapse (d, c) is equivalent to *swapping* edge (a, b) , as shown on the right. In general, swapping an edge replaces it with a set of faces triangulating its opposite chain of edges. We restrict our re-triangulations to those that can be obtained with such a split-collapse combination.

4.4 Weighting Scheme

Considering our central goal is to satisfy the condition $d(X) = 0$, we want $d(X)$ to dominate the objective function in terms of its gradient. As input meshes can present a very wide range of quality $f(X)$, we therefore adapt μ and ν accordingly. We pick $\nu = \nu_t / ||\nabla f(X)||^2$ and $\mu = \mu_t / ||\nabla d(X)||^2$, where ν_t and μ_t are user-defined target gradient norm, by default 1 and 100, respectively. Performing this computation, referred to as `INITIALISEWEIGHTS`, represents the *outer loop* of our algorithm. In a typical penalty method fashion, we have an *inner loop* where we subsequently gradually increase μ until we reach convergence for Problem (3). Remeshing is performed at the end of each outer and inner iteration, as described in Section 4.5.3.

4.5 Remeshing

4.5.1 Operators. We use three local operators: edge split, edge collapse, and edge swap (Figure 6). We only permit operations that do not modify the boundary of the mesh, so as to simplify the use of the resulting map in downstream applications. The edge split has a degree of freedom: the position of the newly introduced vertex, both in $\mathcal{M}$ and $\mathcal{M}_C$. It is initially placed in the center of the edge and then adjusted separately (through a small local optimisation problem) to maximise the quality of its incident tetrahedra.

4.5.2 Quality Vectors. Each of these edge operators replaces a set of cells with another, and to measure the change of quality, we sort the cells' quality values (before and after) into *quality vectors* [Klingner and Shewchuk 2007]. These are then compared *lexicographically*, in the sense that improving the 3rd worst cell (for instance) is considered a global improvement, even if the worst and second worst cells did not get better. We add a 1% margin in the per-cell values comparison, to tolerate a slight (in particular also numerical) decrease of the worst cells' quality, if it allows for a significant improvement of the next one in the quality vector.

For edge splits and edge swaps, the initial quality vectors include all the cells incident to the target edge, and for edge collapses, they include all cells incident to the two end vertices of the edge. As for the quality vector *after* the operation, we simply gather all the new cells it creates. Operations are only *legal* if they maintain positive

element orientation (both in domain and codomain), and either (a) they improve codomain quality or (b) the subsequent quality is above a given threshold. The exact threshold will depend on the remeshing strategy currently being used, as described in the next section. These conditions for validity form a pseudo-hill-climbing approach, inspired by [Wicke et al. 2010].

4.5.3 Strategies. We employ two strategies for remeshing: “Quick Remeshing”, used after each inner iteration to remove the worst elements, and “Deep Remeshing”, performed at the end of each outer iteration to globally improve quality as much as possible. The former strategy uses a priority queue, where we try to remove each tetrahedral cell (sorted from worst to best) by successively splitting, swapping, or collapsing its 6 edges. Once an operation has been successfully performed, (thus improving local quality), the newly-created cells are added to the priority queue. We stop once no operation could be performed, or the worst cell’s quality is above a threshold, which we set to 10^{-4} . This value is to be seen in relation to the $q_{\min}$ value we picked for Problem (2). The goal here is to focus on cells whose quality is close to the bound, which we argue are the main reason for the deformation stalling.

Experiments showed that many but not all low-quality cells can always be removed in this local manner. This leads us to the Deep Remeshing strategy, which performs all edge splits, then all swaps, and then all collapses, wherever it improves quality, throughout the mesh. This is repeated, stopping if the worst cell’s quality is above 10^{-1} , or 5 passes are exceeded. Between passes, vertex positions are optimised for $f(X)$. The high computational cost of this global strategy is the reason we use it only at the end of an outer loop iteration in case the deformation did not succeed yet.

4.5.4 Pre-Remeshing. Before starting the optimisation, we first split all chord edges and faces, i.e. those that are interior but whose vertices lie all on the boundary. Then, one Deep Remeshing pass is performed, followed by edge collapses as long as they improve quality, reducing the number of degrees of freedom so as to aid in reducing the subsequent runtime. The recreation of chords is forbidden in these operations.

4.6 Advancing Front Snapping

As the deformation is a numerical process, the boundary will usually not end up in exactly the target shape. Therefore (and to increase efficiency) we add a snapping mechanism. Snapping is not performed in terms of angles and lengths, but directly in terms of positions, to ensure a perfect match. Concretely, boundary vertices are eventually snapped to their target positions.

This needs to be done with care, though, to not defeat the underlying non-positional nature of the deformation approach. For instance, greedy snapping based on a proximity threshold can cause unescapable traps, as illustrated in Figure 7 right. This is avoided by strictly keeping the snapped region connected.

Concretely, once the deformation has reached a state such that the boundary is globally near its target shape—all boundary edge dihedral angles differ from their target angle by less than 5° —we snap the vertices of a single boundary face to their target positions.

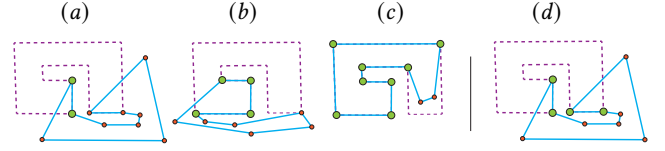


Fig. 7. (a), (b), (c) In this 2D example, we illustrate how we snap vertices (with linear constraints) by conquering them in an advancing-front manner. (d) Snapping in multiple non-connected regions can create situations where no continuous validity-maintaining deformation to the target shape is possible, for reasons similar to Figure 2 right.

As $d(X)$ is invariant to rotation and translation, this is done after applying the global rigid transformation the best aligns that face with its target state (`OPTIMALRIGIDTRANSFORMATION()` in Algorithm 1). This seed face is selected as the one whose neighbourhood (5 rings) has the lowest average residual in $d(X)$. Snapped vertices henceforth are fixed in place in the optimisation problem by eliminating their variables.

Subsequently, in each inner iteration of the optimisation, adjacent vertices are incrementally snapped as well, in an advancing front manner as illustrated in Figure 7, as far as possible without violating any of the guards in Problem (2). An advancing front procedure has been used in the context of tetrahedral deformation before [Mandad et al. 2022], albeit in a very different setting, without a prescribed target shape and directional rather than positional snapping. Importantly, as listed in Algorithm 1, when a vertex is snapped, the virtual vertices of its incident boundary edges (forming the set $\mathcal{E}(S)$) are re-initialised in the codomain, as the snapping may invert the virtual elements.

For simplicity we assume that the mesh has a single boundary component, i.e. no internal cavities, though this could conceptually be lifted with a multi-front approach.

4.7 Progress Metric

This key component is used to decide both when to perform a deeper remeshing pass (i.e. when to move on to the next outer iteration), and when to detect that the algorithm got (numerically) stuck. We consider that progress was made in an iteration if the relative decrease in $d(X)$ is above a user-defined threshold, or if any vertex was newly snapped. This latter criterion comes from the observation that even if the snapping of some vertices has not decreased or even increased $d(X)$, it often helps in making subsequent progress, essentially by nudging the optimiser in the right direction.

Regarding the decrease threshold, we use 10% for the inner loop, and 50% for the outer loop. These two conditions are checked at the end of each inner and outer iteration, w.r.t. to the value of $d(X)$ at the start of the iteration, as summarised in Algorithm 1.

5 Evaluation

Following our discussion on the pros and cons of state-of-the-art tetrahedral mapping methods from Section 2, we strive to appropriately compare our work with said methods. Representative competitors are: Foldover-Free (FOF) [Garanzha et al. 2021], Total-Lifted Content (TLC) [Du et al. 2020], Shrink-and-Expand (SAE) [Nigolian et al. 2024] and Galaxy Maps (GM) [Hinderink and Campen 2023].

Algorithm 1 Differential Advancing Front Mapping**Input:**

$\mathcal{M}$, a tetrahedral mesh with a connected boundary
 ∂_C , prescribed boundary conditions

Output:

$\mathcal{M}'$, a remeshed version of the domain mesh $\mathcal{M}$
 $\mathcal{M}_C$, the same mesh as $\mathcal{M}'$, but satisfying the boundary conditions ∂_C

```

1: procedure DEFORMATIONADVANCINGFRONTMAPPING( $\mathcal{M}, \partial_C$ )
2:   PREREMESHING() //Section 4.5.4
3:   INITIALISEVIRTUALVERTICES( $\partial_C$ ) //Section 4.1
4:    $C \leftarrow ()$  //set of conquered vertices
5:   snap_phase  $\leftarrow$  False
6:    $i \leftarrow 0$ 
7:   while  $C \neq \partial V$  do
8:      $(v, \mu) \leftarrow$  INITIALISEWEIGHTS() //Section 4.4
9:      $j \leftarrow 0$ 
10:    while True do
11:      OPTIMISE Problem (3)
12:      QUICKREMESHING() //Section 4.5.3
13:      if !snap_phase  $\wedge$  MAXDIHEDRALANGLEDIFF()  $< 5^\circ$  then
14:        OPTIMALRIGIDTRANSFORMATION()
15:        snap_phase  $\leftarrow$  True
16:      end if
17:      if snap_phase then
18:         $S \leftarrow$  SNAPNEXTBOUNDARYVERTICES() //Section 4.6
19:         $C \leftarrow C \cup S$ 
20:        INITIALISEVIRTUALVERTICES( $\mathcal{E}(S)$ ) //Section 4.1
21:      end if
22:      if  $j \wedge$  INNERLOOPPROGRESS()  $< 10\%$  then //Section 4.7
23:        break;
24:      end if
25:       $j++$ 
26:       $\mu \leftarrow 10\mu$ 
27:    end while
28:    if  $i \wedge$  OUTERLOOPPROGRESS()  $< 50\%$  then //Section 4.7
29:      fail
30:    end if
31:     $i++$ 
32:    DEEPREMESHING() //Section 4.5.3
33:  end while
34: end procedure

```

We produce a diverse selection of datasets which will serve as basis for highlighting the differences of our Differential Advancing Front Mapping (DAF) algorithm relative to the aforementioned methods in different scenarios. Although our comparisons will focus on ball-topology meshes, often mapped to star-shaped codomains, we emphasise that this is due to the availability of data and the restrictions of some of the other methods, not due to a limitation of our method.

Definition of Success: Especially SAE and GM are known for unpredictable runtimes, due to possibly intense refinement. We hence set a general time limit of 24h per input instance across methods. While SAE and GM come with theoretical success guarantees, we here consider a method to be **practically successful** on an input instance only if the result is produced *within 24h* and it is still valid after truncation to double-precision *floating-point* representation.

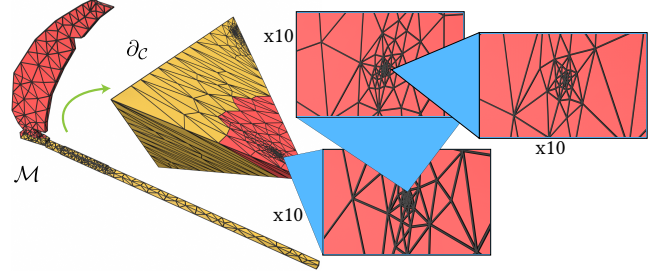


Fig. 8. In the dataset from [Nigolian et al. 2023], connectivity-wise embeddings of a tetrahedron are used to generate challenging boundary maps. Surface patches are then mapped onto planar tetrahedral faces using Tutte embeddings. Poor or random tetrahedral embeddings often leads to large portions of the surface being mapped onto relatively small areas, such as the part highlighted in red. As the x10 zoom-ins show, edge lengths then decrease exponentially, which makes these mapping cases numerically very challenging. The maximum per-boundary-triangle Symmetric Dirichlet Energy recorded for this dataset is 10^{54} .

Table 1. Our method shows the highest practical success rate among optimisation-based methods when dealing with a subset of 3004 mapping cases taken from [Nigolian et al. 2023]. In addition, our practical success rate is similar to that of the constructive method SAE, but requires significantly less mesh refinement, as discussed in Section 5.8.1.

	DAF (ours)	TLC	FOF	SAE
success rate ("SAE Tets")	72.8%	47.0%	15.5%	74.3%

5.1 Distortion Robustness

In order to evaluate our method on numerically difficult cases, we use a subset of the dataset introduced by Nigolian et al. [2023]. It consists of 3004 ball-topology meshes, mapped onto four distinct target shapes. One of these shapes is the so-called "Tet"; the boundary maps onto it were constructed from a per-side Tutte embedding and often present extremely high distortion, as illustrated in Figure 8. We will refer to this dataset as the "SAE Tets". We summarise the practical success rate results in Table 1. Although we are unable to reach a higher practical success rate than the constructive SAE method, superiority over other optimisation-based methods showcases our method's robustness to extreme boundary conditions.

5.2 Knotted Holes

We further test the geometrical limits of our method by taking a difficult case, the famously non-shellable "Knotted Hole" or "Furch's ball" [Furch 1924]. This mesh is constructed by taking a regular block of voxels, and "digging" a hole by sequentially removing voxels, while maintaining its ball topology. By digging 10 voxel segments in a specific fashion, we create a hole representing a knot (in the colloquial sense), as illustrated by Figure 9. The 10 intermediate states of the digging represent an interesting small dataset meant to test how far we can dig, before a mapping method fails to produce a bijective map. As target shape we use a cube (see Figure 9).

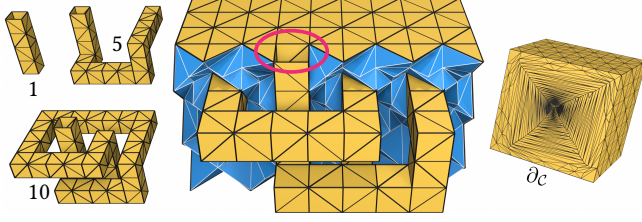


Fig. 9. On the left, surface meshes representing the 1st, 5th and 10th segments dug into a tetrahedralised voxel grid. At the 10th segment, we produce the famous Knotted Hole mesh, as shown in the center (the magenta circle highlights the “entrance” of the hole). This ball-topology mesh is mapped onto a cubic codomain ∂C (right), where the entire hole is flattened onto one of the cube faces.

Table 2. For each method, we evaluate how far we can dig into the Knotted Hole voxel grid before it stops being practically successful (cf. Figure 9). Our method is the only one that can directly map every stage. The (remeshed) domain meshes produced by our method are subsequently used as a second set of inputs for our competitors (columns marked with a [†]), to make the task easier and in particular ensure fixed-mesh feasibility.

	DAF (ours)	TLC	TLC [†]	FOF	FOF [†]	SAE	SAE [†]	GM	GM [†]
max. dig depth	10	4	10	5	4	5	7	4	6

Previous methods may fail in this experiment because there potentially is no solution at all when treating the input mesh as fixed. We thus perform a second experiment, giving the other methods a remeshed version as input for which it is known that there is a solution without further refinement: the output domain mesh of our method. For both experiments, Table 2 illustrates how our method outperforms former methods on these conceptually challenging cases. To the best of our knowledge, our method is the first one capable of automatically producing an explicit bijective map of the Knotted Hole to some simple codomain.

Table 3. Practical success rate in generating a map for the cuboids from [Brückler et al. 2022]. Our method clearly outperforms the state of the art. The numbers for TLC and FOF were taken directly from [Hinderink and Campen 2023].

	DAF (ours)	TLC	FOF	SAE	GM
success rate (“Cuboids”)	96.7%	58.3%	46.7%	18.3%	85%

5.3 Hex-Meshing

Among the many applications of tetrahedral maps, we now focus on hexahedral (hex-) meshing. As dataset we use the 60 challenging mapping instances to *cuboids*¹ from a block-structured hex-meshing context [Brückler et al. 2022]. Given that robustness is a main open challenge in that context, we focus our comparison (see Table 3) on the practical success rate, the ability to generate a bijective map,

¹<https://github.com/HendrikBrueckler/CuboidBlocks>

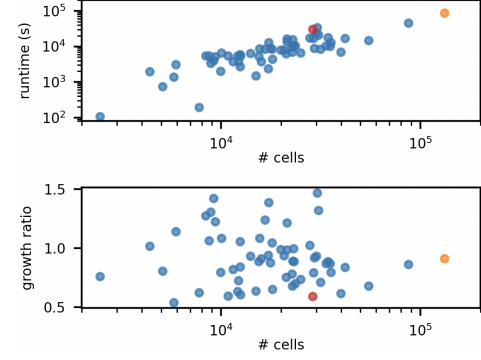


Fig. 10. Runtimes and *growth ratios* (the output #cells divided by the input #cells) for our method for the cuboids dataset. The red dot represents a case where our algorithm got numerically stuck (cf. Section 4.7), and the orange dot represents a timeout after 24h. The reasonable mesh size growth ratios are a good indicator that our method is suitable for downstream use.

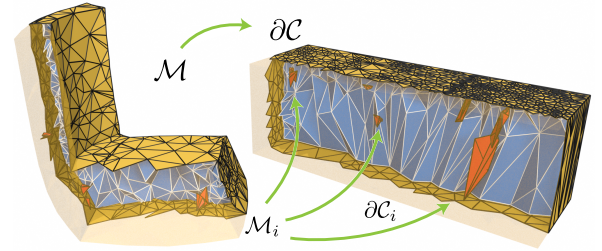


Fig. 11. In the Galaxy Maps (GM) approach [Hinderink and Campen 2023], star-shaped regions are grown around flipped elements (here shown in red) of an initially invalid map (e.g. from a Tutte embedding). Each star is then considered as a separate codomain target ∂C_i , for the corresponding domain submeshes M_i . This can significantly reduce the problem sizes. However, the stars can be very distorted, representing numerically challenging cases.

regardless of efficiency or quality—which can be dealt with subsequently [Brückler and Campen 2023; Kohler et al. 2025]. Nonetheless, Figure 10 illustrates that our method needs not completely sacrifice efficiency to reach its high success rate. From all our experiments, on this hex-meshing-focused dataset the biggest benefit relative to state-of-the-art methods can be observed.

5.4 Galaxy Maps Integration

To reduce runtime, we now turn to the Galaxy Maps approach [Hinderink and Campen 2023], in which a mapping problem is divided into small, localised mapping sub-problems (cf. Figure 11). From the 60 cuboid mapping instances from [Brückler et al. 2022], 1784 subproblems are extracted (henceforth referred to as the “Cuboid Stars” dataset) and can be mapped individually by any method. In the original paper, they use the foliation-based technique introduced in [Campen et al. 2016], which we now add as FOL to our list of competitors for this particular experiment. The results are summarised by Table 4, and Figure 12 gives an overview of the runtime and mesh size growth ratio distribution for all methods.

Table 4. Among the 1784 stars generated by [Hinderink and Campen 2023], valid maps for 1517 of them can be generated by all 5 competitor methods. While our method boasts the highest practical success rate, it is significantly slower than its competitors on these cases, which can represent a relevant limitation depending on the applications.

	success rate	time / input tet (s)		mesh growth ratio		
		avg	max	min	avg	max
DAF (ours)	99.9%	$5.4 \cdot 10^{-1}$	$2.8 \cdot 10^2$	0.5	1.2	224.1
FOF	92.2%	$2.1 \cdot 10^{-3}$	$5.3 \cdot 10^{-2}$	1	1	1
TLC	91.3%	$1.9 \cdot 10^{-3}$	$3.8 \cdot 10^{-2}$	1	1	1
FOL	99.5%	$1.3 \cdot 10^{-3}$	$1.0 \cdot 10^{-1}$	1.0	1.4	35.9
SAE	99.4%	$4.7 \cdot 10^{-4}$	$1.6 \cdot 10^{-3}$	1.0	1.0	1.5

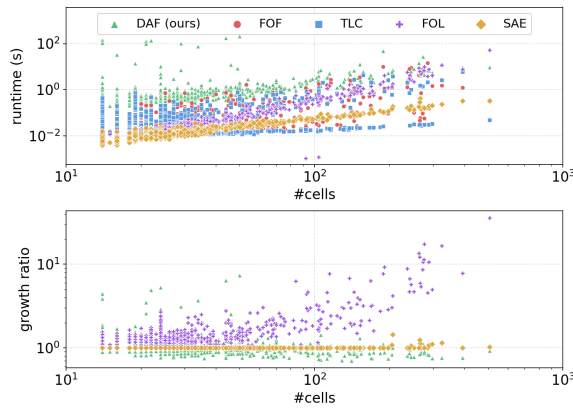


Fig. 12. Runtime and growth ratio distribution of each method on the Cuboid Stars dataset, for the 1517 common success cases. The robustness of our method comes at the price of high runtimes, with outliers taking hours even for small meshes. We used constant bound parameter settings (cf. Section 4) throughout our evaluation. While our method can be orders of magnitude slower than the state of the art, additional experiments showed that performance for these small meshes can be drastically improved by setting more permissive quality bounds in the optimisation problem.

5.5 Higher Genus

We complement our evaluation with topologically non-trivial anatomical mapping cases. MRI scans of human spines are segmented into (overlapping) “functional spine units” (FSUs), each consisting of two adjacent vertebrae, connected through the intervertebral disc between them. For the two (partial) spines (from two distinct patients) depicted in Figure 13 this yields 26 FSUs, each represented as a mesh (one thereof discarded due to defects in the original data), each of genus 2. For anatomical analysis purposes, and to combinatorially transfer volumetric segmentation information, bijective maps between consecutive FSUs, intra-patient and inter-patient, are of interest. This yields a dataset with 117 FSU pairs. Boundary maps were initialised using non-rigid Iterative Closest-Point (nICP) registration. The mapping method performance comparison on this dataset is reported in Table 5. Here, SAE and GM are excluded as they are incompatible with non-star-shaped codomains.

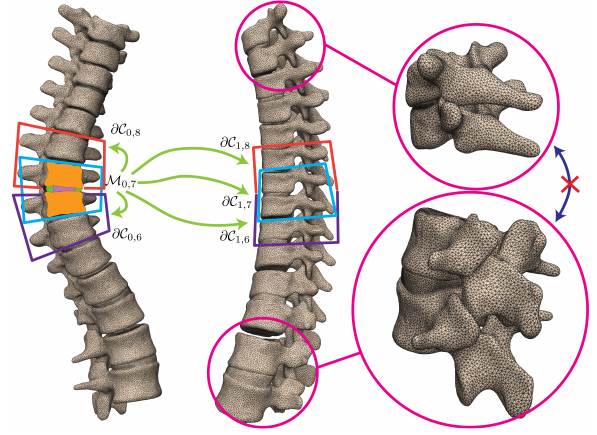


Fig. 13. The “Vertebrae” dataset consists of 117 mapping instances between a total of 25 Functional Spine Unit (FSU) meshes from 2 scoliosis patients (left and center). Only maps between neighbouring FSUs are sought; distant FSUs can be geometrically very different (right), hampering already the determination of a boundary map. This dataset illustrates an application of 3D mapping: transfer of anatomical volumetric segmentations, with bone (orange), intervertebral annulus (green) and nucleus (pink), between FSUs.

Table 5. 86 out of the 117 Vertebrae FSU cases can be mapped by all three methods, while only our method can map them all. Similarly to the Cuboid Stars, this high success rate comes at a significant runtime cost. These two datasets largely differ on mesh size ($\sim 10^2$ vs. $\sim 10^5$ cells on average) and element distortion ($\sim 10^6$ vs. $\sim 10^1$ average distortion), which might explain why the performance gap is less pronounced here.

	success rate	time / input tet (s)		growth ratio		
		avg	max	min	avg	max
DAF (ours)	100%	$1.3 \cdot 10^{-2}$	$2.1 \cdot 10^{-2}$	0.18	0.33	0.44
FOF	94.9%	$8.6 \cdot 10^{-4}$	$1.2 \cdot 10^{-2}$	1	1	1
TLC	78.6%	$3.9 \cdot 10^{-3}$	$5.3 \cdot 10^{-3}$	1	1	1

5.6 Comparisons Summary

In addition to the datasets introduced in the previous sections, we complete our evaluation with meshes taken from the VOLMAP dataset [Cherchi and Livesu 2023]. Specifically, we take the 25 cases from their ‘G1’ subset (initially coming from [Fu et al. 2015]), and the 1924 cases from their ‘G3’ subset (initially taken from [Zhou and Jacobson 2016]). These 1949 ball-topology meshes are all mapped to star-shaped codomains, with boundary maps of significantly lower distortion, compared to the SAE Tets. Regarding the possible codomains that VOLMAP offers, we limit ourselves to the “Stars”, as they are likely to be the most challenging cases compared to the other options, which include simple shapes such as cubes, spheres or octahedra. We will refer to this dataset as the “VOLMAP Stars”. Since most of the domain meshes in this dataset are the same as in the SAE Tets dataset, i.e. only the codomain shapes and boundary maps differ, we avoid a full re-evaluation and restrict this part of the comparison to FOF and TLC.

Table 6. Our method performs better in terms of practical success rate, compared to the state-of-the-art, for most datasets considered. For details on why some comparisons are omitted, we refer the reader to the dedicated sections. Note that for Cuboid Stars, the value reported for GM is actually for FOL [Campen et al. 2016] (the method used internally by GM per star). For the VOLMAP dataset, notice that even with chord pre-splitting (VOLMAP Stars*), TLC and FOF very often fail to produce a bijective map.

Dataset	Section	#meshes	sizes (#tetrahedra)	practical success rate				
				DAF (ours)	TLC	FOF	SAE	GM
SAE Tets	Section 5.1	3004	$[1.5 \cdot 10^2, 1.3 \cdot 10^6]$	72.8%	47.0%	15.5%	74.3%	—
Knotted Holes	Section 5.2	10	$[1.9 \cdot 10^3, 4.5 \cdot 10^3]$	100%	40%	50%	50%	40%
Motorcycle Cuboids	Section 5.3	60	$[2.4 \cdot 10^3, 1.3 \cdot 10^5]$	96.7%	58.3%	46.7%	18.3%	85%
Cuboid Stars	Section 5.4	1784	$[1.2 \cdot 10^1, 2.2 \cdot 10^4]$	99.9%	91.3%	92.2%	99.4%	99.5%
Vertebrae	Section 5.5	117	$[8.3 \cdot 10^4, 3.7 \cdot 10^5]$	100%	78.6%	94.9%	—	—
VOLMAP Stars	Section 5.6	1949	$[2.3 \cdot 10^3, 1.3 \cdot 10^6]$	77.6%	0.7%	0.3%	—	—
VOLMAP Stars*					21.1%	6.8%	—	—
Total		6924	$[12, 1\,283\,094]$					

The overwhelming majority of these mapping instances have no fixed-mesh solution, typically due to some tetrahedra with all four vertices on the boundary being forced into a coplanar state by the prescribed boundary map. To give the fixed-mesh methods a better chance, we perform a second experiment where we pre-split all interior faces and edges whose vertices are all on the boundary (so-called “chord elements”). Success rates for the VOLMAP Stars (both with and without chord splitting) are shown in Table 6, along with a summary of all the datasets involved in our evaluation.

5.7 Position-Driven Deformation Approach

To demonstrate the benefit of our approach based on virtual elements, over a more direct position-driven deformation-based approach, we consider a prototypical version thereof:

$$\begin{aligned}
 \min_X \quad & f(X) \\
 & - \lambda_v \sum_{\tau \in T} \log(\text{vol}(\tau) - v_{\min}) \\
 & - \lambda_q \sum_{\tau \in T} \log(\text{qual}(\tau) - q_{\min}) \\
 & - \lambda_l \sum_{e \in E} \log(\text{len}(e) - l_{\min}) \\
 \text{s.t.} \quad & X = X_C,
 \end{aligned} \tag{4}$$

where X_C are the prescribed boundary positions forming ∂C . The constraint $X = X_C$ can be taken into account again as a penalty term or be addressed, e.g., using an infeasible-start Newton’s method. Similarly to our formulation (2), a feasible solution to (4) would translate into a valid map to C .

A simple, remeshing-enabled algorithm to solve this problem follows naturally:

- (1) Rigidly transform $\mathcal{M}$ to minimise $\|X - X_C\|_\infty$
- (2) Solve Problem 4
- (3) If $\|X - X_C\|_\infty \leq \epsilon \rightarrow$ success
- (4) If no constraint satisfaction progress $\rightarrow$ failure
- (5) Run a Deep Remeshing pass (cf. Section 4.5.3)
- (6) Go to Step (2)

We observe such an approach to often lead to locking boundary vertex trajectories; even a strict $\|X - X_C\|_\infty$ decrease does not mean

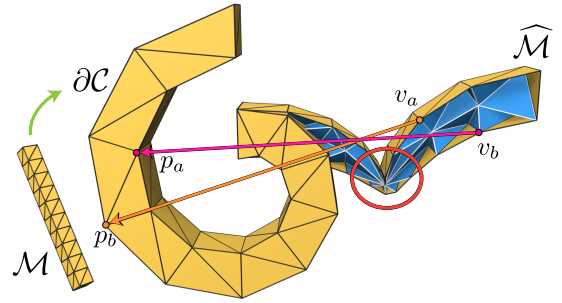


Fig. 14. The rod mesh $\mathcal{M}$ is mapped onto the spiral codomain C with the positional-constraints-based approach (Section 5.7). At the intermediate stage depicted, per-vertex directions reducing their individual constraints violation are incompatible (e.g. v_a and v_b trying to move towards p_a and p_b). This results in the mesh being “stuck” in a local constraints violation minimum; further progress would require either increasing constraints violation, or inverting interior tetrahedra. We emphasise that no remeshing could resolve this conflict, and that the current DOFs of $\widehat{\mathcal{M}}$ are sufficient to represent a valid map. We also note that, due to the absence of virtual elements, dihedral angles may exceed 2π , leading to self-intersections in the red-circled area. Figure 1 right illustrates a similar case.

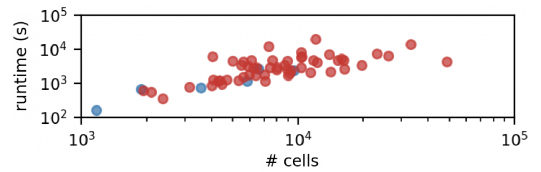


Fig. 15. Runtimes for the position driven deformation algorithm of Section 5.7, for the cuboids from [Brückler et al. 2022]. Blue dots are success cases (10%), and red are failures, where the deformation converged to an invalid situation (cf. Figure 14). Comparing this to Figure 10, the benefit of employing our virtual element approach (96.7% success) becomes clear.

the mesh is deforming in an adequate direction. An example is provided by Figure 14, reminiscent of our discussion from Figure 2. The plot in Figure 15, covering a larger set of instances, further supports the importance of our virtual elements.

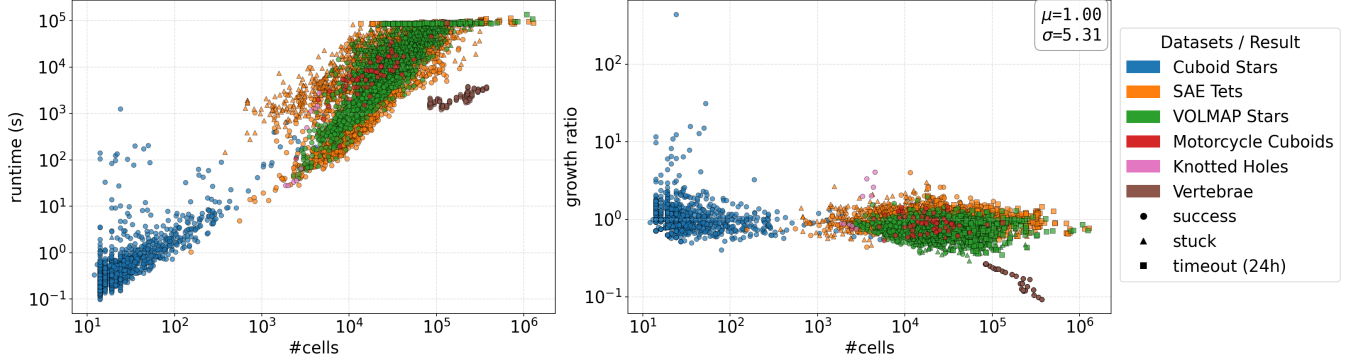


Fig. 16. Every marker on both these plots represent a single mapping pair processed by our method ($n = 6825$). Here we distinguish cases where it failed because no further progress could be made, the so-called “stuck” cases (cf. Section 4.7), from cases where it reached the time limit of 24h. On the left, we see the runtime as a function of the input size in terms of the number of cells. The plot on the right depicts the growth ratio required for our maps to be generated.

5.8 Extended Algorithm Analysis

5.8.1 Overall Performance. Figure 16 further illustrates our algorithm’s performance, both in terms of runtime and refinement. It clearly suggests that mesh size is not a good runtime predictor. Although some cases require significant refinement, this behavior decreases with mesh size. This is an indication that the heavy refinement is necessary to achieve element quality respecting the bound, particularly so for the very small (and distorted) meshes from the Cuboid Stars dataset. Moreover, this represents a significant improvement over refinement-based methods such as SAE and GM. Our method may even decrease mesh size, mostly by collapsing bad quality or low volume elements. Interestingly, among the 6825 total instances tested, only 3705 (~54%) required any remeshing during deformation, past the pre-remeshing step.

We further analyse our algorithm’s runtime with the ternary plots in Figure 17. Overall, instances seem to be balanced between pre-remeshing, on-the-fly remeshing, and optimisation. The latter rarely represents less than 20% of the total runtime, which is to be expected, as it is the core of our method. We note the many cases where pre-remeshing takes a significant part of the runtime, with no on-the-fly remeshing. These are meshes where initially improving minimum quality is sufficient to generate the DOFs necessary to generate a valid map. As mentioned above, these tend to be the smaller meshes from the Cuboid Stars dataset, although a considerable number of larger meshes do not require on-the-fly remeshing either. The runtime-per-cell colouring on the right-hand plot suggests that relatively less pre-remeshing time correlates with longer runtimes.

5.8.2 Remeshing Performance. Figure 18 shows the impact of our remeshing strategies on the average and minimum quality for all meshes. Clearly, it performs very well across the board, often allowing to remove even the worst elements. The average improvement is also significant, as shown on the left-side subplot. These results indicate the soundness of the greedy strategies, showing the concrete benefits of their heavy computational cost. Furthermore, they can serve as a comparison baseline for future remeshing strategies.

On the other hand, Figure 19 provides more insights into how runtime is distributed between the different local edge operations

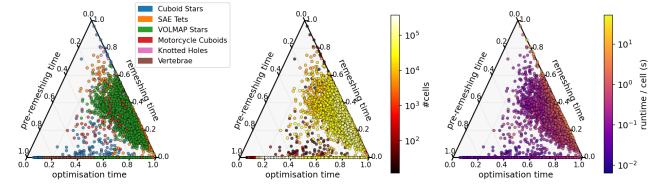


Fig. 17. A barycentric ternary plot representing how runtime is distributed between the pre-remeshing step, and the interleaved remeshing and optimisation steps, for cases where the algorithm succeeded. Plots are coloured (from left to right) by dataset, by mesh size (number of input cells) and runtime per cell.

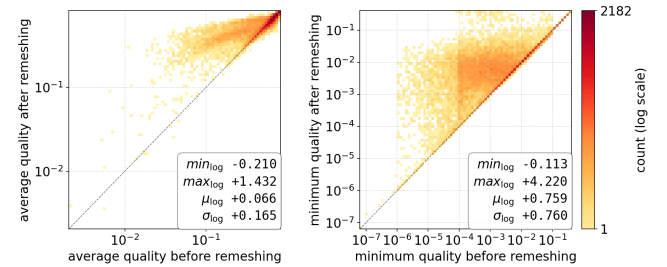


Fig. 18. Heatmap of the quality before and after each run of our remeshing, for all the 3705 meshes that triggered on-the-fly remeshing ($n = 16097$), in log scale. The darker pattern on the right side plot stopping at 10^{-4} is explained by our “Quick Remeshing” strategy only triggering at this specific threshold (cf. Section 4.5.3).

(cf. Section 4.5.1). Swaps dominate the ratio of operations performed, representing ~40% of the operations for the vast majority of meshes. The rest is quite equally distributed between splits and collapses. Interestingly, longer remeshing runtimes are closer to a perfect split-collapse equilibrium. Considering that our quality vector comparison prevents cycling operations (cf. Section 4.5.2), our interpretation is that it still allows for numerous small incremental improvements of the quality. This suggests that imposing stricter quality improvement thresholds for operations could be a path to reducing runtime.

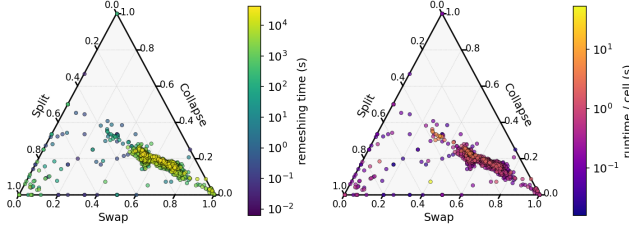


Fig. 19. This shows how remeshing is distributed among the three types of operations we use in our method, and how it relates to the total remeshing time (left) and the overall processing time (right).

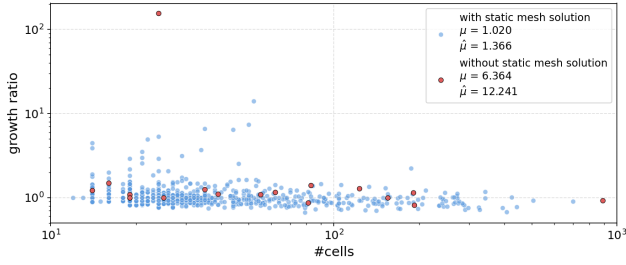


Fig. 20. Among the 1784 Cuboid Stars instances, 1755 (blue) are known to be solvable without altering the input mesh (because fixed-mesh methods like FOF or TLC succeed on them), but for 17 (red) remeshing may be inevitable. Our method generally performs remeshing, not rarely also increasing the mesh size where this is not strictly necessary. Reported are μ , the average mesh size growth ratio, and $\bar{\mu}$, the average of cases with $\mu > 1$. Notice that growth rates are on average significantly lower for the blue cases, suggesting that our method, while not making only minimal use of remeshing, effectively adapts the amount of refinement to the hardness of the instance.

Regarding the overall necessity of remeshing in our method, we first refer to Figure 20. It indicates that the amount of remeshing (and in particular refinement) performed is partially superfluous, yet not excessive: only in 20.6% of the Cuboid Stars instances known to have a fixed-mesh solution the mesh is actually refined (rather than just connectivity-adjusted or internally coarsened). This highlights how our method depends on having not only *sufficient* DOFs, but *adequate* DOFs. Moreover, our method can still perform well without remeshing, as illustrated by Table 7. Table 8 further compares our method's performance when stripped from remeshing over all the datasets considered. This shows that, depending on the type of mapping problem, remeshing can be essential to our method's high success rate. Interestingly, though, even without it, leaving only the virtual elements as key ingredient, our method's performance can, in certain cases, be better than the state of the art's.

5.8.3 Input Quality & Distortion. We can garner interesting insights from Figure 21, showing how our resulting meshes fare against the stability guards discussed in Section 3.2. On the successes side, it is clear that the quality (volume-length) being part of the objective function allows it to reach acceptable values, with the majority of cases in the $[10^{-3}, 10^{-1}]$ range. Having no such term (aside from

Table 7. Success rate of our method (DAF), when disabling on-the-fly remeshing ($\text{DAF}^\dagger$) or all remeshing operations ($\text{DAF}^{\dagger\dagger}$), over the Cuboid Stars with known fixed-mesh solution. Runtimes and growth ratios are reported over the 1740 instances on which all methods succeeded.

	success rate	time / input tet (s)		growth ratio		
		avg	max	min	avg	max
DAF (ours)	99.9%	$6.0 \cdot 10^{-2}$	10.1	0.66	1.02	14.0
$\text{DAF}^\dagger$	99.5%	$2.6 \cdot 10^{-2}$	$2.4 \cdot 10^{-1}$	0.64	0.98	1.85
$\text{DAF}^{\dagger\dagger}$	99.1%	$1.8 \cdot 10^{-2}$	$4.2 \cdot 10^{-1}$	1	1	1
FOF	93.7%	$6.3 \cdot 10^{-3}$	$1.6 \cdot 10^{-1}$	1	1	1
TLC	92.8%	$2.0 \cdot 10^{-3}$	$9.6 \cdot 10^{-2}$	1	1	1

Table 8. Disabling remeshing significantly decreases the success rate for the most challenging datasets. The low difference between $\text{DAF}^\dagger$ and $\text{DAF}^{\dagger\dagger}$ compared to DAF is a strong indicator that on-the-fly remeshing is a particularly essential aspect of the overall remeshing strategy.

	DAF	$\text{DAF}^\dagger$	$\text{DAF}^{\dagger\dagger}$
SAE Tets	72.8%	33.9%	37.6%
Knotted Holes	100%	60%	40%
Motorcycle Cuboids	96.7%	76.7%	41.7%
Cuboid Stars	99.9%	99.5%	99.1%
VOLMAP Stars	77.6%	34.7%	34.4%
Vertebrae	100%	100%	100%

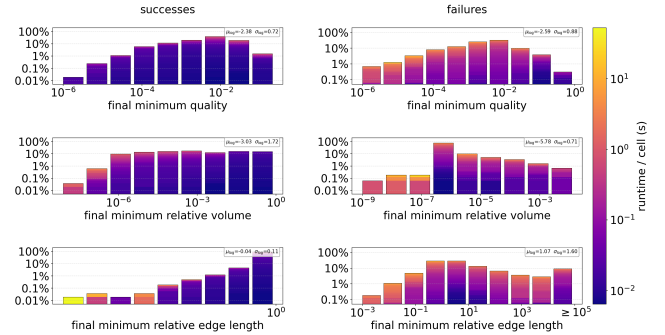


Fig. 21. These histograms summarise the three key metrics we identified as being the main sources of instability (see Section 3.2), split between cases where a map was successfully generated (left), and when our algorithm either got stuck or timed out (right). Each bar is coloured so as to represent how the runtime per cell distributed over each individual quality range.

the log-barrier term) for the volume, the minima uniformly spread across the feasible range. However, there seems to be a strong correlation between the minimum volume and failure, with near 90% of failure cases terminating with a volume very close to the bound. The biggest discrepancy between success and failure clearly comes from the edge length (relative to the minimum codomain boundary length). Indeed, meshes from the SAE Tets and VOLMAP Stars dataset often present extremely small edges on the codomain boundary (cf. Figure 8), which can make it very difficult, or even impossible, to conform to the boundary conditions while staying within our

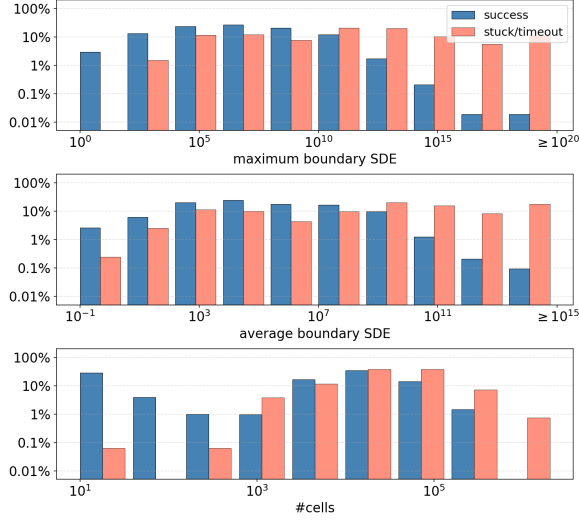


Fig. 22. Overlapping histograms of the success and failures cases, w.r.t. input mesh size, and input boundary map distortion. Map distortion is measured with the 2D Symmetric Dirichlet Energy (SDE). Aside from input size, boundary distortion is a significant predictor of our method's performance.

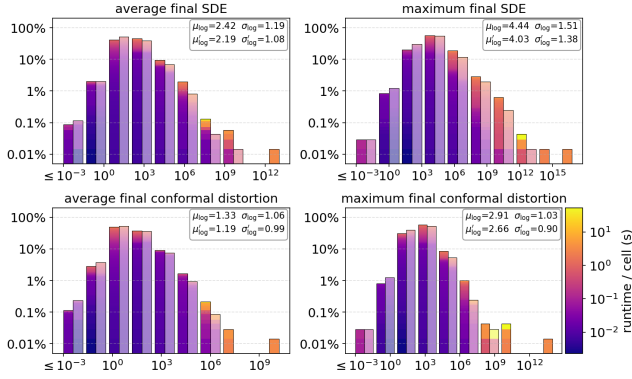


Fig. 23. These histograms summarise the per-mesh maximum and average values for two distortion metrics: the Symmetric Dirichlet Energy (SDE), and the *conformal distortion* $d_{conf} = \text{tr}(J^T J) / (\det J)^{\frac{2}{3}}$. The semi-transparent bars represent the values after optimising these distortions once valid maps have been generated. Bars are coloured based on the runtime per cell of their corresponding cases.

stability guards. These last observations together suggest that our choice for $v_{\min}$ and $l_{\min}$ are not entirely adapted to these two datasets, where almost all failures come from. This is further supported by Figure 22, where the failure case distributions are skewed towards higher input boundary map distortion.

5.8.4 Output Map Distortion. Map distortion is an important aspect for many applications. We thus evaluate our mapping method's performance in that regard, see Figure 23. Both distortion metrics are rather high, which is not surprising, considering that our quality metric focuses on mesh quality rather than map quality. Minimising

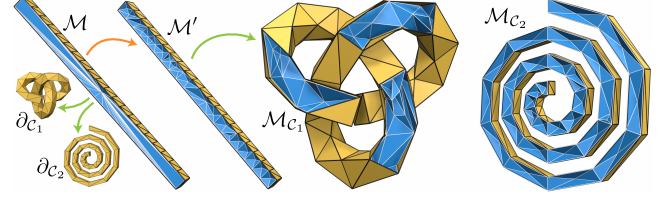


Fig. 24. Mapping the cuboid mesh $\mathcal{M}$ with a single interior vertex to non-star-shaped codomains ∂C_i unaltered is impossible. However, our remeshing scheme generates the DOFs necessary to generate a valid map, between the remeshed domain mesh $\mathcal{M}'$ and codomain meshes $\mathcal{M}_{C_i}$, a knot and a spiral. These two highly-non-star-shaped examples showcase how our remesher is rather conservative in terms of additional DOFs, producing relatively high quality tessellations. Maps were also successfully generated for longer spirals (with up to 30 turns).

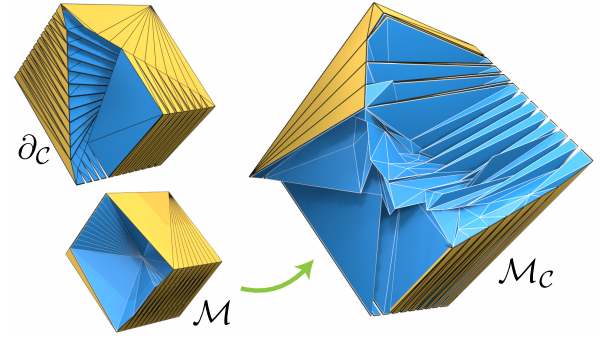


Fig. 25. The domain cube $\mathcal{M}$ is meshed in a naive manner with a single interior vertex. It gets mapped onto a Chazelle polyhedron [Chazelle 1984], famously known for not permitting a tetrahedralisation with few (or even no) interior vertices. Our method automatically refines the mesh as needed (to 290 interior vertices) to arrive at a bijective map onto (and thereby also a tetrahedralisation of) this complex shape.

these distortions after producing a valid map can improve on this to some extent. For better results, one could apply remeshing again, but using map distortion [Abulnaga et al. 2023] as quality metric instead.

5.8.5 Mesh Generation. In case no input tetrahedral mesh is given, just the domain boundary triangle mesh, with our method one does not need to worry about generating a good quality tetrahedralisation to start with for the purpose of computing a map. In particular, if the domain is convex or star-shaped, one can very easily generate a mesh $\mathcal{M}$ with a single interior vertex. Our method's built-in remeshing will automatically generate DOFs as needed. This is illustrated by examples in Figure 24, Figure 25, and Figure 26.

Note that, due to the symmetry of the problem of *bijective* mapping, one may also switch roles and instead of the domain $\mathcal{D}$ initially mesh $\mathcal{C}$. This can be particularly useful if $\mathcal{C}$ is simpler, cf. Figure 26. This symmetry can also be exploited as speed-up: One can independently attempt to generate both $\phi : \mathcal{D} \rightarrow \mathcal{C}$ and $\phi^{-1} : \mathcal{C} \rightarrow \mathcal{D}$ in parallel, and output whichever finishes first.

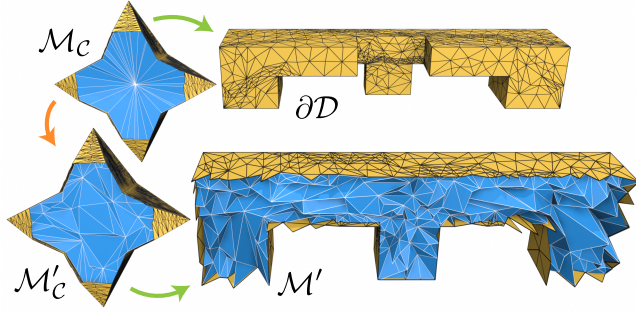


Fig. 26. Instead of mapping a mesh M to a codomain C , one can generate a codomain mesh M_C and inversely map it to domain D . Using a minimalist codomain mesh, e.g. with a single interior vertex for star-shaped boundaries, the mesh generation effort is delegated to our remeshing, which will on-the-fly generate the DOFs needed to yield a mesh that is valid in both domain and codomain (M' and M'_C). This is an example of the versatility of our remeshing, going beyond bare validity, generating a good quality mesh.

6 Limitations & Future Work

The modular structure of our method offers a number of paths for future work, alleviating efficiency limitations and further increasing robustness, so as to harness the full potential of the approach.

Localised Deformation: Perhaps the most obvious of these improvements would be to follow the footsteps of [Hinderink and Campen 2023], applying the proposed mapping strategy locally to faulty regions of a (cheaply constructed, non-injective) initial map. A benefit of our method in that context is that it is not limited to star-shaped regions, which could have a highly positive effect on limiting the size of regions to treat. Considering that the input mesh size is one of the main factors for runtime, this add-on would undoubtedly be interesting.

Remeshing: Although our approach, discussed in Section 4.5.3, does partially prioritise low-quality regions for remeshing, it does not guarantee making subsequent vertex movement possible, nor does it use vertex movement as a criterion to decide where and how to remesh. Here, one could take inspiration from Jin et al. [2014], who specifically focus on a single vertex’s movement (although in 2D) and how to guarantee that it can reach a target position.

Another interesting approach would be to extend the set of candidate operations, and add the ability to chain operations before deciding if they are valid improvements or not. However, doing so would require the use of smart rollback mechanisms to efficiently undo the operation chains. We also note a potential fall-back remeshing strategy consisting of uniformly subdividing the mesh with the so-called “8-tet split” [Plaza and Rivara 2003]. This approach should always generate the DOFs necessary to enable further deformation, but comes at a major computational cost for future steps.

Control Parameters: Parameters such as $q_{\min}$, $v_{\min}$ and $l_{\min}$ from Sections 3.2 and 4.3, or the v_t and μ_t from Section 4.4 have been chosen empirically, to maximise the success rate over all datasets. Using dataset or instance specific parameters proved to enable yielding better results. For instance, decreasing $v_{\min}$ to 10^{-9} yielded a

100% success rate for the Cuboid Stars dataset, but reduced it for all others. On the Cuboids dataset, for each instance we could find parameter settings producing a valid map, but no common setting yielding a valid map for all of them. Looking into ways for automatic parameter adjustment based on the input characteristics, or even within an instance in a local manner, especially in regions where high distortion is necessitated by the prescribed boundary map (cf. Section 5.8.3), would certainly be worthwhile.

Self-Overlapping Codomains: Mapping onto codomains that are self-overlapping [Weber and Zorin 2014] is of interest for certain use cases, including parametrisation-based 3D hexahedral mesh generation [Nieser et al. 2011], analogous to overlapping domains in the context of 2D quadrilateral mesh generation [Zhou et al. 2020]. The sought maps then are not expected to be bijective but locally injective. In such scenarios the codomain’s (inner) dihedral angles can exceed 2π . While we have not explicitly covered this case, it would essentially require only a small extension of our virtual elements construction in Section 4.1: Instead of a wheel, the virtual elements need to close a “multi-wheel”, winding around the boundary edge more than once, and compatibly forming the same number of windings also in the domain space.

7 Conclusion

In this work, we introduced a novel framework for tetrahedral mapping built on deformation, relying on so-called “virtual elements” and on adaptation of the piecewise linear structure through local remeshing operations. Both components turn out to be crucial. Although its runtime performance was empirically shown to not always be on par with the state of the art, it proved to be significantly more successful practically in challenging scenarios, such as mapping instances with difficult boundary conditions or high implied distortion.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 10004346 (Integer-Grid Maps for Mesh Generation and 3D Reconstruction), and the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (AlgoHex, grant agreement No 853343). Special thanks to Xingyi Du for his support in making modifications to his TLC code, to Martin Heistermann for generating the Chazelle Polyhedron mesh, and the CBE lab from the University of Bern for providing the FSU meshes.

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.
- Hendrik Brückler and Marcel Campen. 2023. Collapsing embedded cell complexes for safer hexahedral meshing. *ACM Trans. Graph.* 42, 6 (2023), 1–24.
- Hendrik Brückler, Ojaswi Gupta, Manish Mandad, and Marcel Campen. 2022. The 3D motorcycle complex for structured volume decomposition. *Computer Graphics Forum* 41, 2 (2022), 221–235.
- Marcel Campen, Cláudio T Silva, and Denis Zorin. 2016. Bijective maps from simplicial foliations. *ACM Trans. Graph.* 35, 4 (2016), 1–15.
- Bernard Chazelle. 1984. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.* 13, 3 (1984), 488–507.
- Gianmarco Cherchi and Marco Livesu. 2023. VOLMAP: A large scale benchmark for volume mappings to simple base domains. *Computer Graphics Forum* 42, 5 (2023).

- Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and robust mesh arrangements using floating-point arithmetic. *ACM Trans. Graph.* 39, 6, Article 250 (2020), 16 pages.
- 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), 120.
- Marion Donyach, David Vanderhaeghe, Loic Barthe, and Mario Botsch. 2013. Adaptive remeshing for real-time mesh deformation. In *Eurographics 2013 Short Papers*.
- Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M Kaufman. 2021. Guaranteed globally injective 3D deformation processing. *ACM Trans. Graph.* 40, 4 (2021).
- Patrick E Farrell, Matthew D Piggott, Christopher C Pain, Gerard J Gorman, and Cian R Wilson. 2009. Conservative interpolation between unstructured meshes via supermesh construction. *Computer methods in applied mechanics and engineering* 198, 33–36 (2009), 2632–2642.
- Zachary Ferguson, Teseo Schneider, Danny Kaufman, and Daniele Panozzo. 2023. In-time step remeshing for contacting elastodynamics. *ACM Trans. Graph.* 42, 4 (2023).
- Xiao-Ming Fu, Yang Liu, and Baining Guo. 2015. Computing locally injective mappings by advanced MIPS. *ACM Trans. Graph.* 34, 4 (2015), 1–12.
- Robert Furch. 1924. Zur Grundlegung der kombinatorischen Topologie. In *Abhandlungen aus dem mathematischen Seminar der Univ. 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.
- Steffen Hinderink, Hendrik Brückler, and Marcel Campen. 2024. Bijective volumetric mapping via star decomposition. *ACM Trans. Graph.* 43, 6 (2024), 1–11.
- Steffen Hinderink and Marcel Campen. 2023. Galaxy maps: Localized foliations for bijective volumetric mapping. *ACM Trans. Graph.* 42, 4 (2023), 1–16.
- Steffen Hinderink and Marcel Campen. 2026. DiskScissors: Cutting Arbitrary-Topology Solids for Bijective Mapping. *Computer Graphics Forum* (2026), e70379.
- 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. *Computer Graphics Forum* 33, 5 (2014), 269–279.
- Bryan Matthew Klingner and Jonathan Richard Shewchuk. 2007. Aggressive tetrahedral mesh improvement. In *Proc. 16th International Meshing Roundtable*. 3–23.
- Tobias Kohler, Martin Heistermann, and David Bommes. 2025. HexHex: Highspeed Extraction of Hexahedral Meshes. *ACM Trans. Graph.* 44, 4 (2025), 1–20.
- Manish Mandad, Ruizhi Chen, David Bommes, and Marcel Campen. 2022. Intrinsic mixed-integer polycubes for hexahedral meshing. *Computer Aided Geometric Design* 94 (2022), 102078.
- Federico Meloni, Gianmarco Cherchi, Riccardo Scateni, Livesu Marco, et al. 2024. To What Extent Are Existing Volume Mapping Algorithms Practically Useful?. In *Italian Chapter Conference 2024 - Smart Tools and Applications in Graphics*. The Eurographics Association, 1–8.
- Alexander Naitzat, Gregory Naitzat, and Yehoshua Y Zeevi. 2021. On inversion-free mapping and distortion minimization. *Journal of Mathematical Imaging and Vision* 63, 8 (2021), 974–1009.
- Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. Cubecover—parameterization of 3d volumes. *Computer Graphics Forum* 30, 5 (2011), 1397–1406.
- Valentin Zénon Nigolian, Marcel Campen, and David Bommes. 2023. Expansion cones: A progressive volumetric mapping framework. *ACM Trans. Graph.* 42, 4 (2023).
- Valentin Zénon 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 (2024), 1–14.
- Matthew Overby, Danny Kaufman, and Rahul Narain. 2021. Globally Injective Geometry Optimization with Non-Injective Steps. *Comput. Graph. Forum* 40, 5 (2021), 111–123.
- VN Parthasarathy, CM Graichen, and AF Hathaway. 1994. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design* 15, 3 (1994), 255–261.
- Ángel Plaza and Maria-Cecilia Rivara. 2003. Mesh Refinement Based on the 8-Tetrahedra Longest-Edge Partition. In *12th International Meshing Roundtable, IMR 2003*. 67–78.
- Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable locally injective mappings. *ACM Trans. Graph.* 36, 4 (2017), 1.
- Johnathan Richard Shewchuk. 1996. Robust adaptive floating-point geometric predicates. In *Proc. 12th annual symposium on Computational geometry*. 141–150.
- Jonathan Richard Shewchuk. 2002. What is a good linear element? Interpolation, conditioning, and quality measures. In *11th Int. Meshing Roundtable, IMR 2002*.
- Jason Smith and Scott Schaefer. 2015. Bijective parameterization with free boundaries. *ACM Trans. Graph.* 34, 4 (2015), 70–1.
- Tommaso Sorgente, Silvia Biasotti, Gianmarco Manzini, and Michela Spagnuolo. 2023. A survey of indicators for mesh quality assessment. 42, 2 (2023), 461–483.
- Robert W Sumner and Jovan Popović. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph.* 23, 3 (2004), 399–405.
- Yuanzhen Wang, Beibei Liu, and Yiying Tong. 2012. Linear surface reconstruction from discrete fundamental forms on triangle meshes. *Comput. Graph. Forum* 31, 8 (2012).
- Ofir Weber and Denis Zorin. 2014. Locally injective parameterization with arbitrary fixed boundaries. *ACM Trans. Graph.* 33, 4 (2014), 1–12.
- Jiahao Wen, Jernej Barbič, and Danny M Kaufman. 2025. Optimal r-Adaptive In-Time step Remeshing for Elastodynamics. *ACM Trans. Graph.* 44, 4 (2025), 1–19.
- 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. *Comput. Graph. Forum* 39, 2 (2020).
- Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* 35, 4, Article 39 (2016), 15 pages.
- Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR]
- Leyi Zhu, Michael Tao, Yixin Hu, Daniele Panozzo, and Denis Zorin. 2026. BijectiveRemesh: Maintaining Bijective Mappings for Data Transfer Across Remeshed Manifolds. arXiv:2605.30744
- Daniel Zint, Zhouyuan Chen, Yifei Zhu, Denis Zorin, Teseo Schneider, and Daniele Panozzo. 2025. Topological Offsets. *ACM Trans. Graph.* 44, 4 (2025), 1–19.

A Mesh Overlay

The map resulting from our method is piecewise linear over the domain $\mathcal{D}$: it is defined on the (remeshed) mesh $\mathcal{M}'$, by its vertices’ domain positions and codomain positions ($\mathcal{M}_C'$). Due to the remeshing operations, the linear pieces do not necessarily correspond to the tetrahedra of the original discretisation $\mathcal{M}$. Note that this freedom is necessary to ensure the existence of a piecewise linear map in general—the original discretisation may not permit one. Depending on the use case scenario, it may be beneficial, though, if the piecewise structure of the map is a pure conforming refinement (rather than a more general remeshing) of the input mesh $\mathcal{M}$.

This can be achieved by further refining the final remeshed mesh $\mathcal{M}'$ to a *common refinement* of $\mathcal{M}$ and $\mathcal{M}'$, also referred to as *overlay* or *supermesh* [Farrell et al. 2009], effectively splitting the linear pieces at the cell boundaries of $\mathcal{M}$. Concretely, the cells of this common refinement are all nonempty pairwise intersections of tetrahedra from $\mathcal{M}$ and $\mathcal{M}'$: $\{\tau \cap \tau' \neq \emptyset \mid \tau \in \mathcal{M}, \tau' \in \mathcal{M}'\}$. These are polyhedral but can be split into tetrahedra if desired. Besides naive construction via pairwise intersection, 3D arrangement computation methods [Cherchi et al. 2020; Zhou et al. 2016] and constrained Delaunay techniques [Farrell et al. 2009] are options.

The codomain positions of the vertices newly introduced in this process are set simply by linear interpolation, preserving the map. The result is a map defined on a mesh $\mathcal{M}''$ such that each tetrahedron of input mesh $\mathcal{M}$ coincides with a group of one or more tetrahedra in $\mathcal{M}''$. This, for instance, significantly simplifies the transfer of attributes or other data possibly defined on tetrahedra, faces, edges, or vertices of the input mesh $\mathcal{M}$ onto its refined version $\mathcal{M}''$, e.g. to then be mapped bijectively to the codomain.

An alternative, in case only pointwise evaluation of the map is required, is the evaluation of the map across the sequence of remeshing operations [Zhu et al. 2026].