Bijective Projection in a Shell

ZHONGSHI JIANG, TESEO SCHNEIDER, DENIS ZORIN, and DANIELE PANOZZO, New York University

We introduce an algorithm to convert a self-intersection free, orientable, and manifold triangle mesh $\mathcal{T}$ into a *generalized prismatic shell* equipped with a bijective projection operator to map $\mathcal{T}$ to a class of discrete surface contained within the shell whose normals satisfy a simple local condition. Properties can be robustly and efficiently transferred between these surfaces using the prismatic layer as a common parametrization domain.

The combination of the prismatic shell construction and corresponding projection operator is a robust building block readily usable in many downstream applications, including the solution of PDEs, displacement maps synthesis, Boolean operations, tetrahedral meshing, geometric textures, and nested cages.

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

Additional Key Words and Phrases: Projection, Bijective Map, Envelope, Mesh Adaptation, Attribute Transfer

ACM Reference Format:
Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective Projection in a Shell. *ACM Trans. Graph.* 39, 6, Article 1 (December 2020), 18 pages. https://doi.org/10.1145/3414685.3417769

1 INTRODUCTION

Triangular meshes are the most popular representation for discrete surfaces, due to their flexibility, efficiency, and direct support in rasterization hardware. Different applications demand different meshes, ranging from extremely coarse for collision proxies, to high-resolution and high-quality for accurate physical simulation. For this reason, the adaptation of a triangle mesh to a specific set of criteria (surface remeshing) is a core building block in geometry processing, graphics, physical simulation, and scientific computing.

In most applications, the triangular mesh is equipped with attributes, such as textures, displacements, physical properties, and boundary conditions (Figure 1). Whenever remeshing is needed, these properties must be transferred on the new mesh, a task which has been extensively studied in the literature and for which robust and generic solutions are still lacking (Section 2). Defining a continuous bijective map, more precisely, a homeomorphism where the inverse is also continuous, between two geometrically close piecewise-linear meshes of the same topology is a difficult problem, even in its basic form, when one of these meshes is obtained by adapting the other in some way (e.g., coarsening, refining, or improving triangle shape).a Common approaches to this problem are Euclidean projection [Jiao and Heath 2004], parametrization

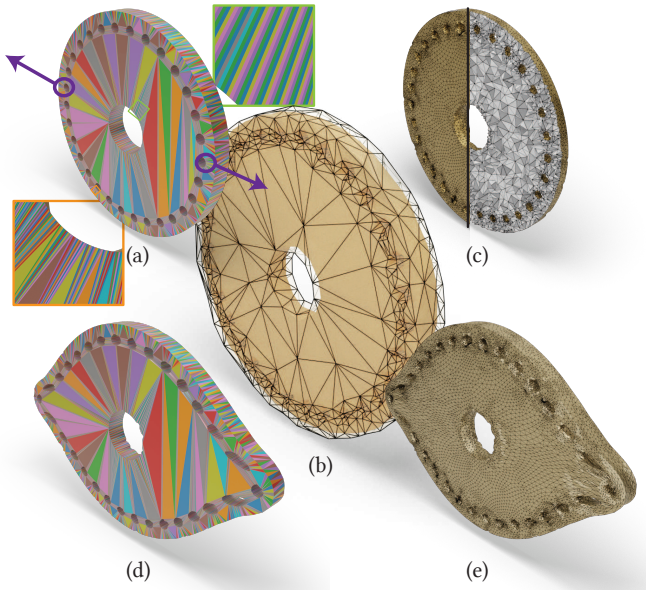


Fig. 1. A low-quality mesh with boundary conditions (a) is remeshed using our shell (b) to maintain a bijection between the input and the remeshed output. The boundary conditions (arrows in (a)) are then transferred to the high-quality surface (c), and a non-linear elastic deformation is computed on a volumetric mesh created with TetGen (e). The solution is finally transferred back to the original geometry (d). Note that in this application setting both surface and volumetric meshing can be hidden from the user, who directly specifies boundary conditions and analyses the result on the input geometry.

on a common domain [Kraevoy and Sheffer 2004; Lee et al. 1998; Praun et al. 2001], functional maps [Ovsjanikov et al. 2012], and generalized barycentric coordinates [Hormann and Sukumar 2017]. However, the problem is not fully solved, as all existing methods, as we discuss in greater detail in Section 2, often fail to achieve bijectivity and/or sufficient quality of the resulting maps when applied to complex geometries. Our focus is on correspondences between meshes obtained during a remeshing procedure, instead of solving the more general problem of processing arbitrary mesh pairs.

In this work, we propose a general construction designed to enable attribute mapping between geometrically close (in a well-defined sense) meshes by jointly constructing: (1) a shell $\mathcal{S}$ around triangle mesh $\mathcal{T}$ spanned by a set of prisms, inducing a volumetric vector field $\mathcal{V}$ in its interior and (2) a projection operator $\mathcal{P}$ that bijectively maps surfaces inside the shell to $\mathcal{T}$, as long as the dot product of the surface face normals and $\mathcal{V}$ is positive (we call such a surface a *section* of $\mathcal{S}$). Given a surface mesh $\mathcal{T}$ and its shell $\mathcal{S}$, it is now possible to exploit the bijection induced by $\mathcal{P}$ in many existing remeshing algorithms by adding to them an additional constraint ensuring that the generated surface is a section of a given shell.

Authors' address: Zhongshi Jiang, jiangzs@nyu.edu; Teseo Schneider, teseo.schneider@nyu.edu; Denis Zorin, dzorin@cs.nyu.edu; Daniele Panozzo, panozzo@nyu.edu, Computer Science Department, New York University, New York, NY.

0730-0301/2020/12-ART1 $15.00
https://doi.org/10.1145/3414685.3417769

As long as the generated mesh is a section, the projection operator $\mathcal{P}$ can be used to transfer application-specific attributes. At a higher level, the middle surface of our shell can be seen as a common parametrization domain shared by sections within the shell: differently from other methods that map the triangle meshes to a disk, region of a plane, a canonical polyhedron, or orbifolds, our construction uses an explicit triangle mesh embedded in ambient space as the common parametrization domain. This provides additional flexibility since it is adaptive to the density of the mesh and naturally handles models with high genus, while being numerically stable under floating-point representation (and exact if evaluated with rational arithmetic). The downside is that it is defined only for sections contained within the shell. The construction and optimization of our shell, and corresponding bijective mapping, is computationally more expensive than remeshing-only methods: our algorithms takes seconds to minutes on small and medium sized models, and might take hours on the large models in our tests.

We evaluate the robustness of the proposed approach by constructing shells for a subset of the models in Thingi10k [Zhou and Jacobson 2016] and in ABC [Koch et al. 2019] (Section 4). We also integrate it in six common geometry processing algorithms to demonstrate its practical applicability (Section 5):

(1) *Proxy*. The creation of a proxy, high-quality remeshed surface to solve PDEs (e.g., to compute geodesic distances or deformations), avoiding the numerical problems caused by a low-quality input in commonly used codes. Bijective projection operators associated with a shell enable us to transfer boundary conditions to the proxy mesh, compute the solution on the proxy, and then transfer the solution back to the original geometry.
(2) *Boolean operations*. The remeshing of intermediate results of Boolean operations, to ensure high-quality intermediate meshes while preserving a bijection to transfer properties between them.
(3) *Displacement Mapping*. The automatic conversion of a dense mesh into a coarse approximation and a regularly sampled displacement height map. Our method generates a bijection that allows us to bake the geometric details in a displacement map.
(4) *Tetrahedral Meshing*. The conversion of a surface mesh of low quality into a high-quality tetrahedral mesh, with bijective correspondence.
(5) *Geometric Textures*. Generation of complex topological structures using volumetric textures mapped to the volumetric parametrization of a simplified shell defined by $\mathcal{P}$. Our analysis on the initial shell also complements the literature on shell maps.
(6) *Nested Cages*. A robust approach to generate a coarse approximation of a surface for collision checking, cage-based deformation, or multigrid approaches.

Our contributions are:

(1) An algorithm to build a prismatic shell and the corresponding projection operator around an orientable, manifold, self-intersection free triangle mesh with arbitrary quality;
(2) A new definition of bijective maps between *close-by* discrete surfaces;
(3) A reusable, reference implementation provided at https://github.com/jiangzhongshi/bijective-projection-shell.

2 RELATED WORKS

We review works in computer graphics spanning both the realization of maps for attribute transfer (Section 2.1), and the explicit generation of boundary cages (Section 2.2), which are closest to our work.

2.1 Attribute Transfer

Transferring attributes is a common task in computer graphics to map colors, normals, or displacements on discrete geometries. The problem is deeply connected with the generation of UV maps, which are piecewise maps that allow to transfer attributes from the planes to surfaces (and composition of a UV map with an inverse may allow transfer between surfaces). We refer to [Floater and Hormann 2005; Hormann et al. 2007; Sheffer et al. 2006] for a complete overview, and we review here only the most relevant works.

Projection. Modifying the normal field of a surface has roots in computer graphics for Phong illumination [Phong 1975], and tessellation [Boubekeur and Alexa 2008]. Orthographic, spherical, and cage based projections are commonly used to transfer attributes, even if they often leads to artifacts, due to their simplicity [Community 2018; Nguyen 2007]. Projections along a continuously-varying normal field has been used to define correspondences between neighbouring surfaces [Ezuz et al. 2019; Kobbelt et al. 1998; Lee et al. 2000; Panozzo et al. 2013], but it is often discontinuous and non-bijective. While the discontinuities are tolerable for certain graphics applications (and they can be reduced by manually editing the cage), these approaches are not usable in cases where the procedure needs to be automated (batch processing of datasets) or when bijectivity is required (e.g., transfer of boundary conditions for finite element simulation). These types of projection may be useful for some remeshing applications to eliminate surface details [Ebke et al. 2014], but it makes these approaches not practical for reliably transferring attributes. Our shell construction, algorithms, and associated projection operator, can be viewed as guaranteed continuous bijective projection along a field.

Common Domains. A different approach to transfer attributes is to map both the source and the target to a common parametrization domain, and to compose the parametrization of the source domain with the inverse parametrization of the target domain to define a map from source to target. In the literature, there are methods that map triangular meshes to disks [Floater 1997; Tutte 1963], region of a plane [Aigerman et al. 2014, 2015; Campen et al. 2016; Fu and Liu 2016; Gotsman and Surazhsky 2001; Jiang et al. 2017; Litke et al. 2005; Maron et al. 2017; Müller et al. 2015; Rabinovich et al. 2017; Schmidt et al. 2019; Schüller et al. 2013; Smith and Schaefer 2015; Surazhsky and Gotsman 2001; Weber and Zorin 2014; Zhang et al. 2005], a canonical coarse polyhedra [Kraevoy and Sheffer 2004; Praun et al. 2001], orbifolds [Aigerman et al. 2017; Aigerman and Lipman 2015,

2016], Poincare disk [Jin et al. 2008; Kharevych et al. 2006; Springborn et al. 2008; Stephenson 2005], spectral basis [Ovsjanikov et al. 2012, 2017; Shoham et al. 2019], and abstract domains [Kraevoy and Sheffer 2004; Pietroni et al. 2010; Schreiner et al. 2004]. While these approaches allow mappings between completely different surfaces, this is a hard problem to tackle in full generality fully automatically, with guarantees on the output (even some instances of the problem of global parametrization, i.e. maps from a specific type of almost everywhere flat domains to surfaces, lack a fully robust automatic solution).

Our approach uses a coarse triangular domain embedded in ambient space as the parametrization domain, and uses a vector field-aligned projection within an envelope to parametrize close-by surfaces bijectively to the coarse triangular domain. Compared to the methods listed above, our approach has both pros and cons. Its limitation is that it can only bijectively map surfaces that are similar to the domain, but on the positive side, it: (1) is efficient to evaluate, (2) guarantees an exact bijection (it is closed under rational computation), (3) works on complex, high-genus models, even with low-quality triangulations, (4) less likely to suffer from high distortion (and the related numerical problems associated with it), often introduced by the above methods. We see our method not as a replacement for the fully general surface-to-surface maps (since it cannot map surfaces with large geometric differences), but as a complement designed to work robustly and automatically for the specific case of close surfaces, which is common in many geometry processing algorithms, as well as serve as a foundation for generating such close surfaces (e.g., surface simplification and improvement, see Section 5)

Attribute Tracking. In the specific context of remeshing or mesh optimization, algorithms have been proposed to explicitly track properties defined on the surface [Cohen et al. 1997; Dunyach et al. 2013; Garland and Heckbert 1997] after every local operation. By following the operations in reverse order, it is possible to resample the attributes defined on the input surface. These methods are algorithm specific, and provide limited control over the distortion introduced in the mapping. Our algorithm provides a generic tool that enables any remeshing technique to obtain such a map with minimal modifications.

2.2 Shell Generation

The generation of shells (boundary layer meshes) around triangle meshes has been studied in graphics and scientific computing.

Envelopes. Explicit [Cohen et al. 1997, 1996] or implicit [Hu et al. 2016] envelopes have been used to control geometric error in planar [Hu et al. 2019a], surface [Cheng et al. 2019; Guéziec 1996; Hu et al. 2017], and volumetric [Hu et al. 2019b, 2018] remeshing algorithms. Our shells can be similarly used to control the geometric error introduced during remeshing, but they offer the advantage of providing a bijection between the two surfaces, enabling to transfer attributes between them without explicit tracking [Cohen et al. 1997]. We show examples of both surface and volumetric remeshing in Section 5. Also, [Bajaj et al. 2002; Barnhill et al. 1992] utilize envelopes for function interpolation and reconstruction, where our optimized shells can be used for similar purposes.

Shell Maps. 2.5D geometric textures, defined around a surface, are commonly used in rendering applications [Chen et al. 2004; Huang et al. 2007; Jin et al. 2019; Lengyel et al. 2001; Peng et al. 2004; Porumbescu et al. 2005; Wang et al. 2003, 2004]. The requirement is to have a thin shell around the surface that can be used to map an explicit mesh copied from a texture, or a volumetric density field used for ray marching. Our shells are naturally equipped with a 2.5D parametrization that can be used for these purposes, and have the advantage of allowing users to generate coarse shells which are efficient to evaluate in real-time. The bijectivity of our map ensures that the volumetric texture is mapped in the shell without discontinuities. We show one example in Section 5.

Boundary Layer. Boundary layers are commonly used in computational fluid dynamics simulations requiring highly anisotropic meshing close to the boundary of objects. Their generation is considered challenging [Aubry et al. 2017, 2015; Garimella and Shephard 2000]. These methods generate shells around a given surface, but do not provide a bijective map suitable for attribute transfer.

Collision and Animation . Converting triangle meshes into coarse cages is useful for many applications in graphics [Sacht et al. 2015], including proxies for collision detection [Calderon and Boubekeur 2017] and animation cages [Thiery et al. 2012]. While not designed for this application, our shells can be computed recursively to create increasingly coarse nested cages. We hypothesize that a bijective map defined between all surfaces of the nested cages could be used to transfer forces from the cages to the object (for collision proxies), or to transfer handle selections (for animation cages). [Botsch and Kobbelt 2003; Botsch et al. 2006] uses a prismatic layer to define volumetric deformation energy, however their prisms are disconnected and only used to measure distortion. Our prisms could be used for a similar purpose since they explicitly tesselate a shell around the input surface.

2.3 Robust Geometry Processing

The closest works, in terms of applications, to our contribution are the recent algorithms enabling black-box geometry processing pipelines to solve PDEs on meshes *in the wild*.

[Dyer et al. 2007; Liu et al. 2015] refines arbitrary triangle meshes to satisfy the Delaunay mesh condition, benefiting the numerical stability of some surface based geometry processing algorithms. These algorithms are orders of magnitude faster than our pipeline, but, since they are refinement methods, cannot coarsen dense input models. While targeting a different application, [Sharp et al. 2019] offers an alternative solution, which is more efficient than the extrinsic techniques [Liu et al. 2015] since it avoids the realization of the extrinsic mesh (thus naturally maintaining the correspondence to the input, but limiting its applicability to non-volumetric problems) and it alleviates the introduction of additional degrees of freedom. [Sharp and Crane 2020] further generalizes [Sharp et al. 2019] to handle non-manifold and non-orientable inputs, which our approach currently does not support.

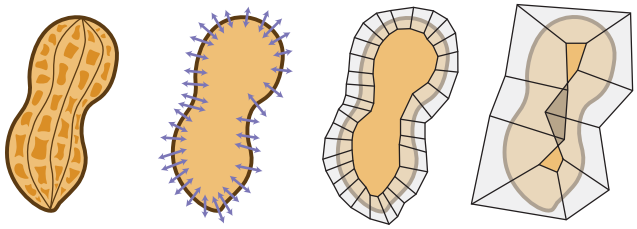


Fig. 2. Overview of our algorithm. We start from a triangle mesh, find directions of extrusion, build the shell, and optimize to simplify it.

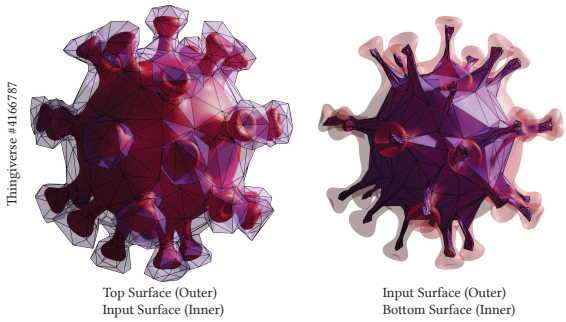


Fig. 3. Example of the top (left, outer) and bottom (right, inner) surface of the prismatic shell.

TetWild [Hu et al. 2019b, 2018] can robustly convert triangle soups into high-quality tetrahedral meshes, suitable for FEM analysis. Their approach does not provide a way to transfer boundary conditions from the input surface to the boundary of the tetrahedral mesh. Our approach, when combined with a tetrahedral mesher that does not modify the boundary, enables to remesh low-quality surface, create a tetrahedral mesh, solve a PDE, and transfer back the solution (Figure 1). However, our method does not support triangle soups, and it is limited to manifold and orientable surfaces.

2.4 Isotopy between surfaces

[Chazal and Cohen-Steiner 2005; Chazal et al. 2010] presents conditions for two sufficiently smooth surfaces to be isotopic. Specifically, the projection operator is a homeomorphism. [Mandad et al. 2015] extends this idea to make an approximation mesh that is isotopic to a region. However, they did not realize a map suitable for transferring attributes.

3 METHOD

Our algorithm (Figure 2) converts a self-intersection free, orientable, manifold triangle mesh $\mathcal{T} = \{V_{\mathcal{T}}, F_{\mathcal{T}}\}$, where $V_{\mathcal{T}}$ are the vertex coordinates and $F_{\mathcal{T}}$ the connectivity of the mesh, into a shell composed of generalized prisms $\mathcal{S} = \{(B_{\mathcal{S}}, M_{\mathcal{S}}, T_{\mathcal{S}}), F_{\mathcal{S}}\}$, where $B_{\mathcal{S}}$, $M_{\mathcal{S}}$, $T_{\mathcal{S}}$ are bottom, middle, and top surfaces of the shell, consisting of bottom, middle, and top triangles of the prisms, and $F_{\mathcal{S}}$ is the connectivity of the prisms (Figure 3). The algorithm initially generates a shell $\mathcal{S}$ whose middle surface $M_{\mathcal{S}}$ has the same geometry as the input surface $\mathcal{T}$ (possibly with refined connectivity), and then optimizes it while ensuring that $\mathcal{T}$ is contained inside and projects bijectively to $M_{\mathcal{S}}$. The shell induces a volumetric vector field $\mathcal{V}$ and a projection operator $\mathcal{P}$ in the interior of each of its prisms (Section 3.1). This output can be used directly in many geometry processing tasks, as we discuss in detail in Section 5.

We first introduce the definition of our projection operator $\mathcal{P}$ and the conditions required for bijectivity of its restrictions to sections of the shell (Section 3.1). We then define shell validity (Section 3.2), present our algorithm for creating an initial shell (Section 3.3) and optimizing it to decrease the number of prisms (Section 3.4). To simplify the exposition, we initially assume that our input triangle mesh does not contain *singular points* (defined in Section 3.3) and boundary vertices, and we explain how to modify the algorithm to account for these cases in sections 3.5 and 3.6.

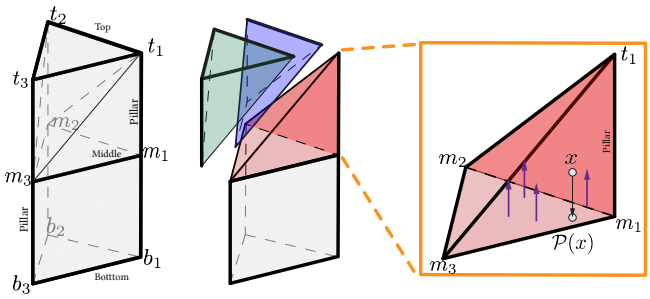


Fig. 4. A prism Δ (left) is decomposed into 6 tetrahedra (middle, for clarity, we only draw the 3 tetrahedra of the top slab). Each tetrahedron has a constant vector field in its interior (pointing toward the top surface), which is parallel to the only pillar of the prism that contains the point.

3.1 Shell and Projection

Let us consider a single generalized prism Δ in a prismatic layer $\mathcal{S}$ (Figure 4 left). The generalized prism Δ is defined by the position of the vertices of three triangles, one at the top, with coordinates t_1, t_2, t_3, one at the bottom, with coordinates b_1, b_2, b_3, and one in the middle, implicitly defined by a per-vertex parameter $\alpha_i \in [0, 1]$, with coordinates $m_i = \alpha_i t_i + (1 - \alpha_i) b_i, i = 1, 2, 3$. We will call the top (bottom) "half" of the prism *top (bottom) slab* (we refer to Appendix E for an explanation on why we need two slabs). For brevity, we will refer to a generalized prism as a prism.

Decomposition in Tetrahedra. We decompose each prism Δ into 6 tetrahedra (3 in the top slab and 3 in the bottom one, Figure 4 middle), using one of the patterns in [Dompierre et al. 1999, Figure 4]. The patterns are identified by the orientation (rising/falling) of the two edges cutting the side faces of the prism. While, for a single prism, any decomposition would be sufficient for our purposes, we need a consistent tetrahedralization between neighboring prisms to avoid inconsistencies in the projection operator. To resolve this ambiguity, we use the technique proposed in [Garimella and Shephard 2000]: we define a total ordering over the vertices of the middle surface of Δ (naturally, we use the vertex id) and split (for each half of the prism) the face connecting vertices v_1 and v_2 with a rising edge if $v_1 < v_2$ and a falling edge otherwise.

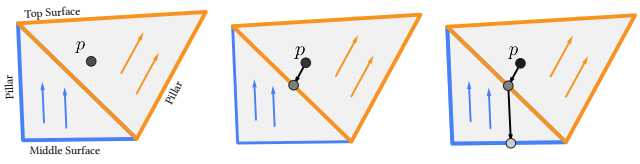


Fig. 5. A point p (left) is traced through $\mathcal{V}$ inside the top part of the shell. A ray with p as origin and $\mathcal{V}$ as direction is cast inside the orange tetrahedron (middle). The procedure is repeated (on the blue tetrahedron) until the ray hits a point in the middle surface (right).

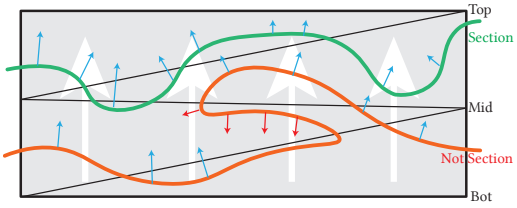


Fig. 6. A 2D illustration for the normal dot product condition. The blue arrows agrees with the background vector field (white arrows), while the red arrows do not agree.

Forward and Inverse Projection. We define a piecewise constant vector field $\mathcal{V}$ inside the decomposed prism, by assigning to each tetrahedron $\boldsymbol{T}_j^\Delta$, $j = 1, \ldots, 6$, the constant vector field defined by the only edge of $\boldsymbol{T}_j^\Delta$ which is a *oriented pillar* of Δ connecting the bottom surface to the top surface passing through the middle surface). That is, for any $p \in \boldsymbol{T}_j^\Delta$

$$\mathcal{V}(p) = t_i - b_i, \tag{1}$$

where i is the index of the vertex corresponding to the pillar edge of $\boldsymbol{T}_j^\Delta$. Note that $\mathcal{V}$ is constant on each tetrahedron and might be discontinuous on the boundary: we formally define the value of $\mathcal{V}$ on the boundary as any of the values of the incident tetrahedra. This choice does not affect our construction. There is exactly one integral (poly-)line passing through each point of the prism if all the decomposed tetrahedra have positive volumes (Theorem 3.2). This allows us to define the *projection operator* $\mathcal{P}(p)$ for a point $p \in \Delta$ as the intersection of the integral line $f_p(t)$ of the vector field $\mathcal{V}$ passing through p, with the middle surface of Δ (Figure 5). Intuitively, we can project any mesh that does not fold in each prism (Figure 6) to the middle surface. Formally, we introduce the following definition, to describe this property in terms of the triangle normals of the mesh.

With a slight abuse of the notation, for meshes and collections of prisms A and B, we use $A \cap B$ to denote the intersection of their corresponding geometry.

Definition 3.1. A *section* $\hat{\mathcal{T}}$ of a prism Δ is a manifold triangle mesh whose intersection with Δ is a simply connected submesh $\hat{\mathcal{T}} \cap \Delta$ whose single boundary loop is contained in the boundary of Δ, excluding its top and bottom surface, and such that for every point $p \in \hat{\mathcal{T}} \cap \Delta$ the dot product between the face normal $n(p)$ and the vector field $\mathcal{V}(p)$ is strictly positive. Similarly, a triangle mesh $\hat{\mathcal{T}}$ is a *section of a shell* $\mathcal{S}$, if it is a section of all the prisms of $\mathcal{S}$.

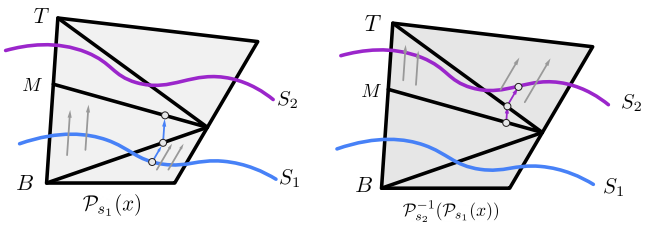


Fig. 7. The composition $\mathcal{P}_{\mathcal{S}_2}^{-1}(\mathcal{P}_{\mathcal{S}_1}(x))$ with $x \in \mathcal{S}_1$, of a direct and an inverse projection operator defines a bijection between two sections $\mathcal{S}_1$ and $\mathcal{S}_2$.

Note that this definition implies that all sections are contained inside the shell. Additionally, the definition implies that the section does not intersect with either bottom or top surface. However, our definition allows for the bottom or top surface to self-intersect. The intersection of the shell does not invalidate the *local* definition of projection since it is defined per prism. Allowing intersections is crucial to an efficient implementation of our algorithm since it allows us to take advantage of a static spatial data structure in later stages of the algorithm (Section 3.4).

Theorem 3.2. *If all 6 tetrahedra $\boldsymbol{T}_j^\Delta$ in a decomposition of a prism Δ have positive volume, then the projection operator $\mathcal{P}$ defines a bijection between any section $\hat{\mathcal{T}}$ of Δ and the middle triangle of the prism (M in Figure 7).*

Proof. We prove this theorem in Appendix A.1. □

The *inverse projection operator* $\mathcal{P}^{-1}$ is defined for a section $\hat{\mathcal{T}}$ as the inverse of the forward projection restricted to $\hat{\mathcal{T}}$. It can be similarly computed by tracing the vector field in the opposite direction, starting from a point in the middle surface of the prism. Note that, differently from the inverse Phong projection [Kobbelt et al. 1998; Panozzo et al. 2013], whose solution depends on the root-finding of a quadric surface, our shell has an explicit form for the inverse and does not require a numerical solve. The combination of forward and inverse projection operators allows to bijectively map between any pair of sections, independently of their connectivity (Figure 7). An interesting property of our forward and inverse projection algorithm, which might be useful for applications requiring a provably bijective map, is that our projection could be evaluated exactly using rational arithmetic.

3.2 Validity Condition

Shell, projection operator, section definitions, and the bijectivity condition (Theorem 3.2) are dependent on a specific tetrahedral decomposition, which depends on vertex numbering.

To ensure that our shell construction is independent from the vertex and face order, we define the validity of a shell by accounting for all 6 possible tetrahedral decompositions [Dompierre et al. 1999, Figure 4].

Definition 3.3. We say that a prismatic shell $\mathcal{S}$ is *valid with respect to a mesh* $\hat{\mathcal{T}}$ if it satisfies two conditions for each prism.

I1 *Positivity.* The volumes of 24 tetrahedra (Appendix A.2) corresponding to 6 tetrahedral decompositions are positive.
I2 *Section.* $\hat{\mathcal{T}}$ is a section of $\mathcal{S}$ for all 6 decompositions.

If a shell is valid, from I1, I2, then by Theorem 3.2, it follows that any map between sections induced by the projection operator $\mathcal{P}$ is bijective.

I2 ensures that the input mesh is a *valid section independently from the decomposition, that is, we require the dot product to be positive with respect to all* three *pillars of a prism inside the convex hull. An interesting and useful side effect of this validity condition is that it ensures the bijectivity of a natural nonlinear parametrization of the prism interior (Appendix D, Figure 25).*

3.3 Shell Initialization

We now introduce an algorithm to compute a *valid* prismatic shell $\mathcal{S} = \{(B_{\mathcal{S}}, M_{\mathcal{S}}, T_{\mathcal{S}}), F_{\mathcal{S}}\}$ with respect to a given triangle mesh $\mathcal{T} = \{V_{\mathcal{T}}, F_{\mathcal{T}}\}$ such that $\mathcal{T}$ is geometrically identical to the middle surface of $\mathcal{S}$. We assume that the faces of the triangle mesh are consistently oriented.

Extrusion Direction. The first step of the algorithm is the computation of an extrusion direction for every vertex of $\mathcal{T}$. These directions are optimized to be pointing towards the *outside* of the triangle mesh (which we assume to be orientable), that is, they must have a positive dot product with the normals of all incident faces. More precisely, for a vertex v, we are looking for a direction d_v such that $d_v \cdot n_f > 0$ for each adjacent face f with normal n_f. We can formulate this as the optimization problem

$$\begin{aligned} &\max_{d_v} \min_{f \in N_v} n_f \cdot d_v, \\ &\text{s.t.} \quad n_f \cdot d_v \geq \epsilon, \qquad \forall f \in N_v \\ &\qquad \|d_v\|^2 = 1. \end{aligned} \tag{2}$$

A solution, if it exists, can be found solving the following quadratic programming problem (Appendix C)

$$\begin{aligned} \min \quad & \|x\|^2 \\ \text{s.t.} \quad & \mathbf{C}x \geq 1, \end{aligned} \tag{3}$$

with $d_v = x/\|x\|$ and $\mathbf{C}$ the matrix whose rows are the normals n_f of the faces in the 1-ring N_v of vertex v. Solutions not satisfying $\|x\| \leq 1/\epsilon$ needs to be discarded (Appendix C). The QP can be solved with an off-the-shelf solver[Cheshmi et al. 2020; Stellato et al. 2017], and in particular it can be solved exactly [Gärtner and Schönherr 2000] to avoid numerical problems. Note that the Problem (2) is studied in a similar formulation in [Aubry and Löhner 2008] but their solution requires tolerances in multiple stages of the algorithm to handle cospherical point configurations.

The admissible set of (2) might be empty for a vertex v, that is, no vector d_v satisfies $\mathbf{C}d_v \geq \epsilon$. In this case we call v a *singularity*. For example, Figure 12 shows a triangle mesh containing a singularity: there exist no direction whose dot product with the adjacent face normals is positive. To simplify the explanation, we assume for the remainder of this section that $\mathcal{T}$ does not contain singularities and also that it does not contain boundaries: we postpone their handling to sections 3.5 and 3.6.

Proposition 3.4. *Let $\mathcal{T}$ be a closed (without boundary) triangle mesh without singularities and $\mathcal{N}$ be a per-vertex displacement field satisfying $\mathbf{C}\mathcal{N}_i > 0$ for every vertex v_i of $\mathcal{T}$. Then there exist a strictly positive per-vertex* thickness *δ_i such that vertices t_i and b_i obtained by displacing v_i by δ_i in the direction of $\mathcal{N}_i$ and in the opposite direction, define a shell that satisfies invariant I1.*

Proof. We provide a proof in Appendix A.2. □

Initial Thickness. We first show that a strictly positive per-vertex thickness δ exists for a shell $\mathcal{S}$ with $\mathcal{T}$ as its middle surface, and then discuss a practical algorithm to realize it.

Theorem 3.5. *Given a closed, orientable, self-intersection free triangle mesh $\mathcal{T}$ such that for all its vertices Problem* (2) *has a solution, a shell $\mathcal{S}$ exists such that $\mathcal{T}$ is the middle surface and there exist a strictly positive per-vertex thickness δ.*

Proof. We provide a proof in Appendix A.3. □

To find a valid per-vertex thickness δ_i to construct the top surface, we initially cast a ray in the direction of $\mathcal{N}$ for each vertex and measure the distance to the first collision with $\mathcal{T}$ (and cap it to a user-defined parameter δ_{max} if no collision is found). An initial top mesh is built with this extrusion thickness; then we test whether any triangle in the top surface intersects $\mathcal{T}$, through triangle-triangle overlap test [Guigue and Devillers 2003], and iteratively shrink δ_i in this triangle by 20% until we find a thickness that prevents intersections between the input and the top surface. Analogously, we build the bottom shell along the opposite direction. Note that the thickness of a vertex for the top and bottom surface can be different.

Validity of the Initial Shell. Proposition 3.4 and Theorem 3.5 ensure that the initial shell constructed using a displacement field $\mathcal{N}$ obtained by solving (3), satisfies property I1. However, $\mathcal{T}$ might not formally satisfy the conditions for being a section of $\mathcal{S}$, despite being identical to the middle surface of $\mathcal{S}$, due to our definition of the projection operator $\mathcal{P}$. The reason for this can be seen in Figure 8. After the initialization, the middle surface is the input mesh. Thus every prism Δ corresponds to a triangle T_Δ of $\mathcal{T}$. The intersection $\Delta \cap \mathcal{T}$, required to check if $\mathcal{T}$ is a section of Δ (Definition 3.1) contains points from both T_Δ and its 1-ring neighborhood.

The projection operator (and thus the definition of the section) is based on all tetrahedral decompositions of Δ; it is possible that multiple tetrahedra (with different vector field values in $\mathcal{V}$) overlap on the boundary of Δ.

Depending on the dihedral angles in the mesh $\mathcal{T}$, it is possible that the dot product between one of the pillars (orange in Figure 8) and the face normals of some of the triangles from 1-ring neighborhood (green in Figure 8) is negative. While it may seem that this problem could be addressed by changing the definition so that each edge is assigned to one of the incident triangles, so that the field direction only in one incident tetrahedron needs to be considered, this problem is more significant than it may seem, as it leads to instability under small perturbations (e.g., due to floating-point rounding of coordinates). Such small perturbations can change the set of triangles intersecting a prism and thus violate the validity of the shell and, consequently, the bijectivity of $\mathcal{P}$. We propose instead

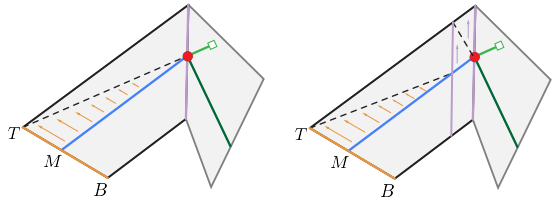


Fig. 8. The vector field aligned with the pillar edge (orange) has a negative dot product with the green triangle normal; the tetrahedron with this field direction meets the green triangle at the red vertex. As a consequence, the mid-surface is not a section. After topological beveling, the shell becomes valid since the dot product between the green normal and the new pillar (purple) is positive.

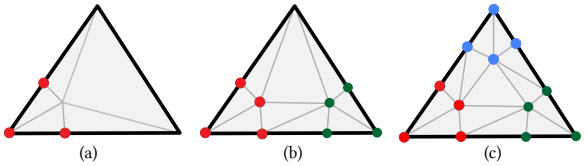


Fig. 9. The beveling patterns used to decompose prisms for which $\mathcal{T}$ is not a section.

to refine $\mathcal{T}$, without changing its geometry, so that the shell corresponding to the refined mesh satisfies I2 (i.e., its middle surface is a section).

Topological Beveling. We identify a prism Δ_v for which I2 does not hold and use a beveling pattern [Conway et al. 2016; Coxeter 1973; Hart 2018] to decompose Δ_v in a way that $\mathcal{T}$ becomes a section for all 6 decompositions (I2). We refer to this operation as *topological beveling*, as it does not change the geometry of the mesh, only its connectivity (Figure 10). We use the pattern in Figure 9a for Δ_v, and we use the other two patterns (b) and (c) on the adjacent prisms to ensure valid mesh connectivity. The positions of the vertices are computed using barycentric coordinates (we used $t = 0.2$, i.e., the orange dot is at 1/5 of the horizontal edge), and the normals of the newly inserted vertices are copied from the closest vertex (in Figure 9, the internal vertices have the normal of the triangle corner with the same color).

Theorem 3.6. *Suppose $\mathcal{T}$ is the middle surface of $\mathcal{S}$, and neither $T_\mathcal{S}$ or $B_\mathcal{S}$ intersects with $\mathcal{T}$. After topological beveling, I2 holds, that is, $\mathcal{T}$ is a section of the shell $\mathcal{S}$ for all 6 decompositions.*

Proof. We provide a proof in Appendix A.4. □

Output. The output of this stage is a valid shell with respect to $\mathcal{T}$ (Section 3.2), that is, it satisfies I1 and I2.

3.4 Shell Optimization

During shell optimization, we perform local operations (Figure 11) on a valid shell to reduce its complexity and increase the quality. Before applying every operation, we check the validity of the operation to ensure that: (1) the resulting middle surface will be manifold [Dey et al. 1999] and (2) the shell will be valid with respect to $\mathcal{T}$ (to ensure a bijective projection). We forbid any operation that does not pass these checks. We would like to remark that, while there are different choices to guide the shell modification, we experimentally discovered that allowing shell simplification and optimization consistently leads to thicker shells with a richer space of sections.

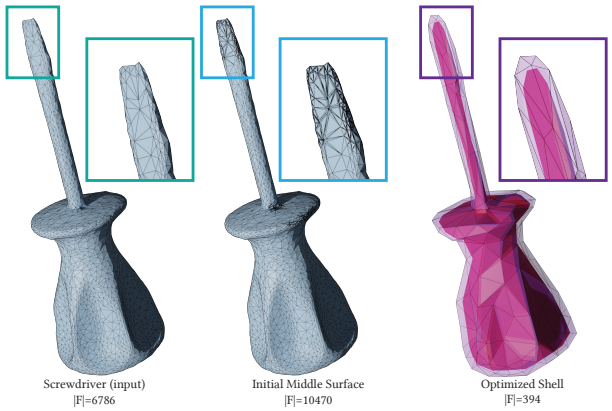


Fig. 10. Our algorithm refines the input model (left) with beveling patterns (middle) to ensure the generation of a valid shell. The subsequent shell optimizations gracefully remove the unnecessary vertices (right).

Theorem 3.7. *Let $\mathcal{S}$ be a valid shell with respect to a mesh $\mathcal{T}$ and let $C = \{\Delta_i\}_{i\in I} \subset \mathcal{S}$ be a collection of prisms such that the middle surface M_C of C is a simply connected topological disk.*

Let $\mathbb{O}$ be an operation replacing C with a new collection of prisms $C' = \{\Delta'_i\}_{i\in I'}$, preserving both geometry and connectivity of the sides of the prism collection C, and ensuring that $M_{C'}$ is a simply connected topological disk.

If these three assumptions hold:

1. *property I1 holds for C',*
2. *the top and bottom surfaces of C' do not intersect $\mathcal{T}$ ($T_{C'} \cap \mathcal{T} = B_{C'} \cap \mathcal{T} = \varnothing$),*
3. *the dot product condition $n(p) \cdot \mathcal{V}(p) > 0$ is satisfied for all points $p \in \mathcal{T} \cap \Delta'_i$ for all pillars of every prism Δ'_i of C',*

then, $\forall i \in I'$, $\mathcal{T} \cap \Delta'_i$ is a simply connected topological disk. In other words, $\mathcal{T}$ is a section of the new shell $\mathcal{S}'$ obtained by applying the operation $\mathbb{O}$ to $\mathcal{S}$.

Proof. We prove this theorem in Appendix A.5. □

We note that assumption (2) in Theorem 3.7 prevents the input surface from crossing the bottom/top surface, thus avoiding it to move in the interior of a region covered by more than one prism.

Our local operations (satisfying the definition of $\mathbb{O}$ in Theorem 3.7) are translated from surface remeshing methods [Dunyach et al. 2013] since our shell can be regarded as a triangle mesh (middle surface) extruded through a displacement field $\mathcal{N}$. All the local operations described below directly change the middle surface, and consequently affect the extruded shell. After every operation, the middle surface is recomputed by intersecting $\mathcal{T}$ with the edges of the prisms in $\mathcal{S}$.

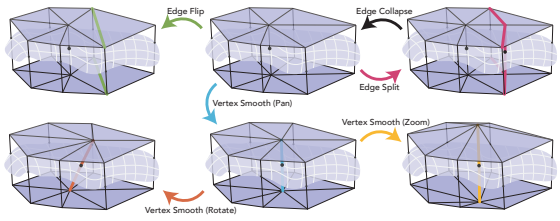


Fig. 11. Different local operations used to optimize the shell. Mesh editing operations translate naturally to the shell setting. For vertex smoothing, we decompose the operation into 3 intermediate steps: pan, zoom, and rotate.

Shell Quality. We measure the quality of the shell $\mathcal{S}$ using the MIPS energy [Hormann and Greiner 2000] of its middle surface $M_{\mathcal{S}}$. For each triangle T of the middle surface, we build a local reference frame, and compute the affine map J_T transforming the triangle into an equilateral reference triangle in the same reference frame. The energy is then measured by

$$\sum_{T \in M_{\mathcal{S}}} \frac{\text{tr}(J_T^T J_T)}{\det(J_T)}.$$

This energy is invariant to scaling, thus allowing the local operations to coarsen the shell whenever possible while encouraging the optimization to create well-shaped triangles. Good quality of the middle surface decreases the chances, for the subsequent operations, to violate the shell invariants.

Shell Connectivity Modifications. We translate three operations for triangular meshes to the shell settings (Figure 11 top). Edge collapse, split, and flip operations can be performed by simultaneously modifying the top and bottom surfaces and retrieve the positions for the middle surface through the intersection. We only accept the operations if they pass the invariant check.

Vertex Smoothing. Due to the additional degree of freedom on vertex-pairs (position, direction, and thickness), we decompose the smoothing operations into three components (Figure 11 bottom). *Pan* moves the positions of the top and bottom vertex at the same time, minimizing the MIPS quality of the middle surface. Neither the thickness or direction will be changed. *Rotate* re-aligns the local direction to be the average of the neighboring ones while keeping the position of the middle vertex fixed. *Zoom* keeps the direction and position of the middle vertex, and set the thickness of both top and bottom to be 1.5 times of the neighbor average, capped by the input target thickness.

Invariant Check. We use exact orientation predicates [Shewchuk 1997] to make sure all the prisms satisfy positivity (I1). Further, we ensure that the original surface $\mathcal{T}$ is not intersecting with the bottom and top surface, except at the prescribed singularities. The check is done using the triangle-triangle overlap test [Guigue and Devillers 2003], accelerated using a static axis-aligned bounding box tree constructed from $\mathcal{T}$. To accelerate the checks for normal condition, for each prism Δ_i, we maintain a list triangles overlapping with its convex hull (an octahedron), and check their respective normals

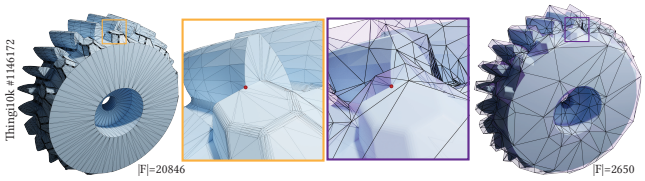


Fig. 12. A model with 48 singularities and a close up around one (left). Our shell is pinched around each of them without affecting other regions (right).

against all the three pillars of Δ_i. These three checks ensure that the three conditions in Theorem 3.7 are satisfied. Note that the vertex smoothing operation is continuous, in the sense that any point between the current position and the optimal one improves the shell. We, however, handle it as a discrete operation to check our conditions: we attempt a full step, and if I1 is not satisfied, we perform a bisection search for a displacement that does. We avoid bisection for the other two conditions since they are expensive to evaluate.

Projection Distortion. An optional invariant to maintain (not necessary for guaranteeing bijectivity, but useful for applications), is a bound on the maximal distortion $\mathcal{D}_{\mathcal{P}}(\Delta)$ of $\mathcal{P}$ for a prism Δ. We measure it as the maximal angle between the normals of the set C containing the faces of $\mathcal{T}$ intersecting Δ and $\mathcal{V}$:

$$\mathcal{D}_{\mathcal{P}}(\Delta) = \max_{p \in C} \angle(n_p, \mathcal{V}(p)),$$

where $\angle$ is the unsigned angle in degrees. This quantity is bounded from below by the smallest dihedral angle of $\mathcal{T}$, making it impossible to control exactly. However, we can prevent it from increasing by measuring it and discarding the operations that increase it. In our experiments, we use a threshold of 89.95 degrees.

Scheduling and Termination. Our optimization algorithm is composed of two nested loops. The outer loop repeats a set of local operations until the face count between two successive iteration decreases by less than 0.01%. In the inner loop we: (1) flip every edge of $\mathcal{S}$ decreasing the MIPS energy and avoiding high and low vertex valences [Dunyach et al. 2013]; (2) smooth all vertices which include pan, zoom, and rotate; and (3) collapse every edge of $\mathcal{S}$ not increasing the MIPS energy over 30. Note that for every operation, we check the invariants, the projection distortion, and manifold preservation and reject any operation violating them. After the outer iteration terminates (i.e. the shell cannot be coarsened anymore), we further optimize the shell with 20 additional iterations of flips and vertex smoothing.

3.5 Singularities

Singularities, i.e. vertices of $\mathcal{T}$ for which the constraints set of problem (2) is empty, are surprisingly common in large datasets (Figure 12 shows an example). For instance, in our subset of Thingi10k [Zhou and Jacobson 2016], although only 0.01% vertices are singular, 8% of the models have at least one singular point. This has been recently observed as a limitation for the construction of nested cages [Sacht et al. 2015, Appendix A], and it is a well-known issue when building boundary layers [Aubry et al. 2017, 2015; Garimella

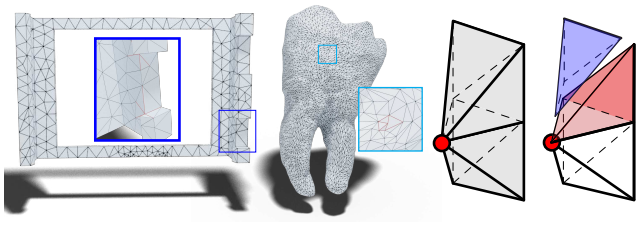


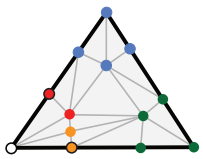
Fig. 13. Left to right: examples of a singularity as a feature point and as a meshing artefact; and illustration of the degenerate prism with a singularity (red) and its tetrahedral decomposition made of only four tetrahedra.

and Shephard 2000]. There are two main situations that give rise to singular points. The first one naturally generates a singular point when more than two ridge-lines meet (e.g., figures 12 and 30), thus making the point a feature point. The second one is a pocket-like mesh artifact, often produced as a result of mesh simplification (Figure 13).

While singularities might seem fixable by applying local smoothing or subdivision as a pre-process, it is not desirable in the case of a feature point, and is likely to introduce self-intersection or more serious geometric inconsistency. Therefore, due to the above reasons and observing that they are very uncommon, we propose to extend our theory (Appendix B) and algorithm to handle isolated singularities by *pinching* the thickness of the shell. Note that in the rare case where two singular points are sharing the same edge, they are *automatically* separated by our topological beveling.

Pinching. We extend our definition of the shell by allowing it to have zero thickness on singularities, thus tessellating the degenerate prism with 4 instead of 6 tetrahedra (Figure 13). We further remark that these isolated points must be excluded from Definition 3.1. In the implementation, this requires to change the intersection predicates to skip the singular vertices. With this change, the singularity becomes a trivial point of the projection operator $\mathcal{P}$, and the rest of our shell can still be used in applications without further changes. Since singularities tends to be isolated (they are usually located at the juncture of multiple sharp features), this solution has minimal effects on applications: for example, when our shell is used for remeshing, pinching the shell at singularities will freeze the corresponding isolated vertices while allowing the rest of the mesh to be freely optimized.

The topological beveling algorithm is changed most significantly: for singularities, there is no pillar to copy from. In this case, we apply an additional edge split, to use the pattern in the inset (with the singularity marked by a white dot) in the one-ring neighborhood of the singularity. The newly inserted vertices lie either inside a triangle (uncircled red and orange dots), or in the interior of an edge (circled red and orange dots). Therefore, we assign to the orange vertices the average normal of the two adjacent triangles, and to red the pillar of the connected orange one.



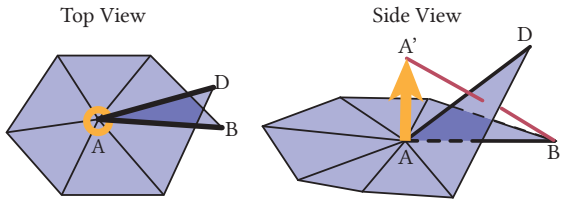


Fig. 14. AA' is a direction with positive dot product with respect to all its neighboring faces. However, no valid shell can be built following that direction.

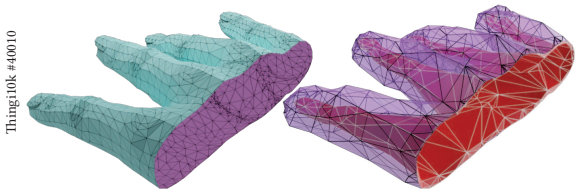


Fig. 15. An example of a mesh with boundary.

Additionally, the edges connecting singularities will always be beveled/split after beveling. Therefore no prism will contain more than one singular point. We discuss the technical extensions for our proofs to shells with pinched prisms in Appendix B.

3.6 Boundaries.

We introduced our algorithm, assuming that the input mesh does not have boundaries. We will now extend our construction to handle this case, which requires minor variations to our algorithm.

For some vertices on the boundary, it might be impossible to extrude a valid shell (Figure 14), even if problem (2) has a solution, as Theorem 3.5 does not apply in its original form. We identify such cases by connecting every edge in the 1-ring neighborhood of the boundary vertex to the extruded point and check if they collide with the existing 1-ring triangles (e.g., the triangle $A'AB$ intersects the existing input triangle in Figure 14). If it is the case, we consider this vertex as a singularity, and we pinch the shell. Note that this is an extremely rare case and, in our experiments, we detected it only for models where the loss of precision in the STL export introduces rounding noise on the boundary.

Once we pinch all boundary singularities, our construction extends naturally to the boundary. The only necessary modification is in the shell optimization (Section 3.4), where we skip all operations acting on boundary vertices to maintain the bijectivity of the induced projection operator (Figure 15). We thus freeze these vertices and never allow them to move or be affected by any other modification of the shell. Note that, in certain applications, it might also be useful to freeze additional non-boundary vertices to ensure that these remain on the middle surface during optimization (e.g., to exactly represent a corner of a CAD model).

4 RESULTS

Our algorithm is implemented in C++ and uses Eigen [Guennebaud et al. 2010] for the linear algebra routines, CGAL [The CGAL Project

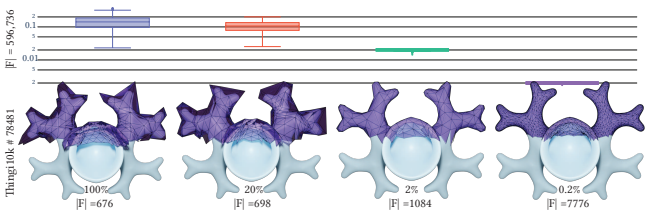


Fig. 16. The effect of different target thickness on the number of prisms |F| of the final shell, and distribution of final thickness (shown as the box plots on the top).

2020] and Geogram [Lévy 2015] for predicates and spatial searching, and libigl [Jacobson et al. 2016] for basic geometry processing routines. We run our experiments on cluster nodes with a Xeon E5-2690 v2 @ 3.00GHz. The reference implementation used to generate the results is attached to the submission and will be released as an open-source project.

Robustness. For each dataset, we selected the subset of meshes satisfying our input assumptions: intersection-free, orientable, manifold triangle meshes without zero area triangles (tested using a numerical tolerance 10^{-16}). We test self-intersections by two criteria: a ball of radius 10^{-10} around each vertex does not contain non-adjacent triangles; and all the dihedral angles are larger than 0.1 degrees.

We tested our algorithm on two datasets: (1) Thingi10k dataset [Zhou and Jacobson 2016] containing, after the filtering due to our input assumptions, 5018 models; and (2) the first chunk of for the ABC dataset [Koch et al. 2019] with 5545 models. The only user-controlled parameter of our algorithm is the target thickness of our shell; in all our experiments (unless stated otherwise), we use 10% of the longest edge of the bounding box. In Figure 16, we show how the target thickness influences the usage of the shell: a thicker shell provides a larger class of sections, thus accommodates more processing algorithms, while a thinner one offers a natural bound on the geometric fidelity of the sections.

Our algorithm successfully creates shells for all 5018 models for Thingi10k and 5545 for ABC. We show a few representative examples of challenging models for both datasets in Figure 17, including models with complicated geometric and topological details. In all cases, our algorithm produces coarse and thick cages, with a bijective projection field defined.

We report as a scatter plot the number of output faces, the timing, and the memory used by our algorithm (Figure 18). In total, the number of prisms generated by our algorithm is 7% and 2% of the number of input triangles for the Thingi10k and ABC dataset respectively and runs with no more than 4.7 GB of RAM. The generation and optimization of the shell takes 5min and 59s in average and up to 8.6 hours for the largest model. 50% of the meshes finish in 3 minutes and 75% in 6 minutes and 15 seconds.

Comparison to Simple Baselines. In Figure 19 we compare to two baseline methods based on [Garland and Heckbert 1998]. For each method, we generate a coarse mesh, uniformly subdivide it for visualization purposes, and query the corresponding spatial position

Fig. 17. Gallery of shells built around models from Thingi10k [Zhou and Jacobson 2016] and ABC [Koch et al. 2019].

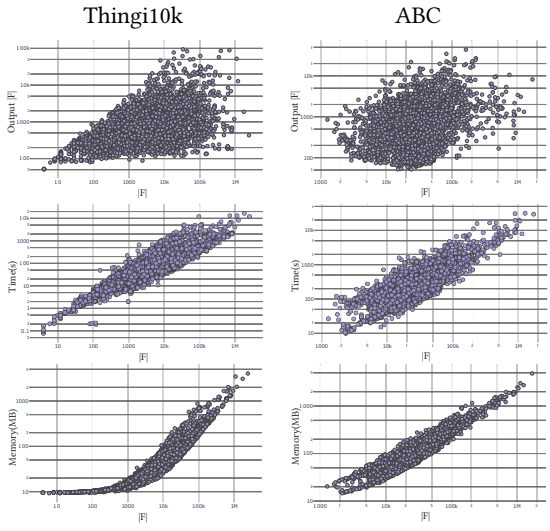


Fig. 18. Statistics of 5018 shells in Thingi10k dataset [Zhou and Jacobson 2016] (left) and 5545 in ABC Dataset [Koch et al. 2019] (right).

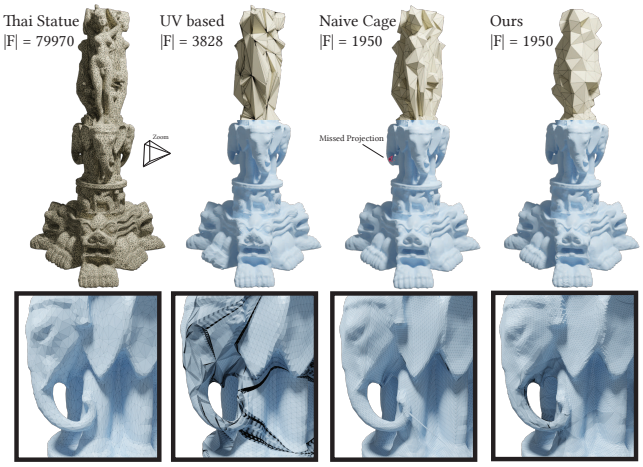


Fig. 19. The *UV based method* cannot simplify the prescribed seams, and introduces self-intersections. The projection induced by the *Naive Cage* method is not continuous and not bijective; it leads to visible spikes in the reconstructed geometry.

on the original input to form the subdivided mesh. The *UV based* method is a conventional way of establishing correspondence in the context of texture mapping. However, the robust generation of a UV atlas satisfying a variety of user constraints is still an open problem. We use the state-of-the-art methods [Jiang et al. 2017; Li et al. 2018] to generate a low-distortion bijective parametrization, and use seam-aware decimation technique [Liu et al. 2017] to generate the coarse mesh. Due to the complex geometry and the length of the seam (Figure 19 second figure), the simplification is not able to proceed beyond the prescribed seam while maintaining the bijectivity, making the pipeline inadequate especially for building computational domains.

We also set up a baseline of the *Naive Cage* method by creating a simplified coarse mesh with [Garland and Heckbert 1998] and use Phong projection to establish the correspondence [Kobbelt et al. 1998; Panozzo et al. 2013]. Such attribute transfer is not guaranteed to be bijective; some face may not be projected (Figure 19 third image). With our method, we can generate a coarse mesh while having a low-distortion bijective projection (Figure 19 fourth image).

Numerical accuracy. To evaluate the numerical error introduced by our projection operator when implemented with floating-point arithmetic, we *transfer* the vertices of the input mesh to the middle surface and *inverse transfer* them from the middle surface back to the input mesh. We measure the Euclidean distance with respect to the source vertices (Figure 20). We compare the same experiment with the Phong projection [Kobbelt et al. 1998]. This alternative approach exhibits distance errors up to 10^{-5} even after ruling out the outliers for which the method fails due to its lack of bijectivity. The maximal error of our projection is on the order of 10^{-8}; this error could be completely eliminated (for applications requiring an exact bijection) by implementing the projection operator and its inverse using rational arithmetic.

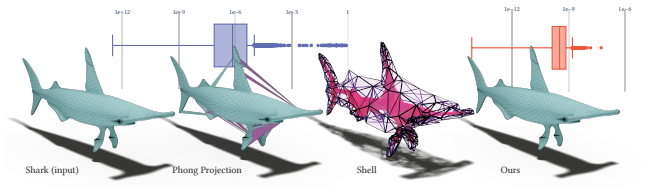


Fig. 20. Our projection is three orders of magnitude more accurate than the baseline method, and bijectively reconstructs the input vertex coordinates.

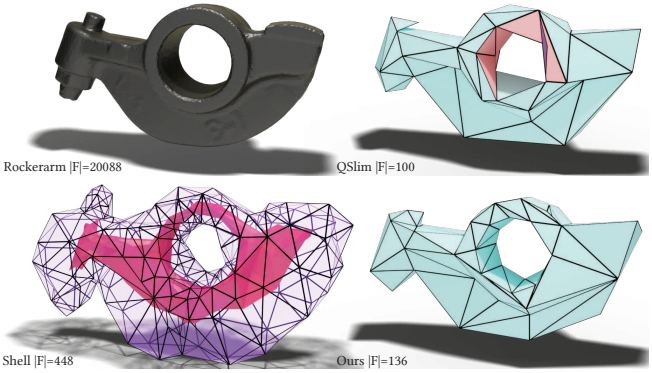


Fig. 21. We attempt to simplify *rockerarm* (top left) from 20088 triangles to 100. QSlim [Garland and Heckbert 1998] succeeds in reaching the target triangle count (top right) but generates an output with a self-intersection (red) and flipped triangles (purple). With our shell constraints (bottom left), the simplification stagnates at 136 triangles, but the output is free from undesirable geometric configurations. Note that both examples use the same quadratic error metric based sceduling [Garland and Heckbert 1998].

5 APPLICATIONS

Using our shell $\mathcal{S}$, we implement the following predicates and functions:

- is_inside(p): returns true if the point $p \in \mathbb{R}^3$ is inside $\mathcal{S}$.
- is_section($\mathcal{T}$): returns true if the triangle mesh $\mathcal{T}$ is a section of $\mathcal{S}$.
- $\mathcal{P}(p)$: returns the prism id (pid), the barycentric coordinates (α, β) in the corresponding triangle of the middle surface, and the relative offset distance from the middle surface (h, which is -1 for the bottom surface, and 1 for the top surface) of the projection of the point p
- $\mathcal{P}^{-1}(\text{pid}, \alpha, \beta, h)$ is the inverse of $\mathcal{P}(p)$.
- $\mathcal{P}_{\mathcal{T}}(\text{tid}, \alpha, \beta) = \mathcal{P}(q)$, where q is the point in the triangle tid of the mesh $\mathcal{T}$, with barycentric coordinates α, β.
- $\mathcal{P}_{\mathcal{T}}^{-1}(\text{pid}, \alpha, \beta)$ is the inverse of $\mathcal{P}_{\mathcal{T}}$.

As explained in Section 3, our shell may self-intersect and we opted to simply exclude the overlapping regions. In practice this affects only the function is_inside(p) which needs to check if p is contained in two or more non-adjacent prisms.

These functions are sufficient to implement all applications below, demonstrating the flexibility of our construction and how easy it is to integrate in existing geometry processing workflows.

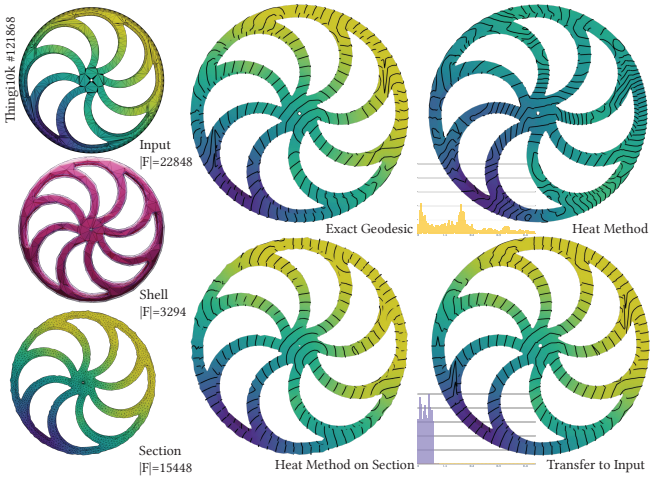


Fig. 22. The heat method (right top) produces inaccurate results due to a poor triangulation (left top). We remesh the input with our method (left bottom), compute the solution of the heat method [Crane et al. 2013] on the high-quality mesh (middle bottom) and transfer the solution back to the input mesh using the bijective projection (bottom right). This process produces a result closer to the exact discrete geodesic distance [Mitchell et al. 1987] (top middle, error shown in the histograms).

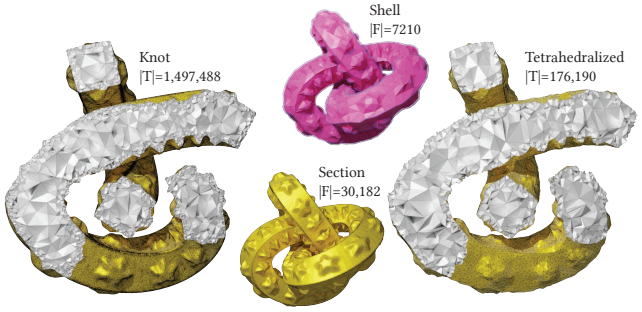


Fig. 23. *Knot* simplified with our method and tetrahedralized with TetGen. The original model is converted to 1,503,428 tetrahedra (left) while the simplified surface is converted to only 176,190 tetrahedra (right).

Remeshing. We integrated our shell in the meshing algorithm proposed in [Dunyach et al. 2013] by adding envelope checks ensuring that the surface is a section after every operation. After simplification, we can use the projection operator to transfer properties between the original and remeshed surface (e.g., figures 22, 23). Since the remeshed surface is a section, a very practical side effect of our construction is that the remeshed surface is *guaranteed to be free of self-intersections*, As shown in Figure 21, the constraints enforced through our shell prevents undesirable geometric configurations (intersections, pockets, or triangle flips).

Proxy. A particularly useful application of our shell is the construction of proxy domains for the solutions of PDEs on low-quality meshes. Additionally, by specifying the target thickness parameter (Figure 16), we are able to bound the geometry approximation

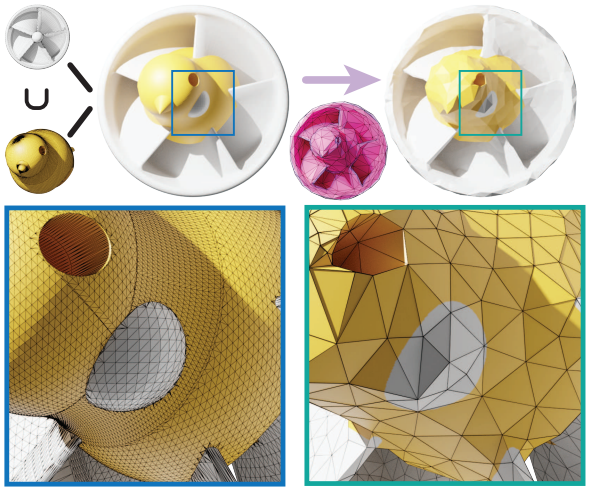


Fig. 24. The union of two meshes is coarsened through our algorithm, while preserving the exact correspondence, as shown through color transfer.

error to the input as well. Using our method, we can (1) convert a low-quality mesh to a proxy mesh with higher quality and desired density, (2) map the boundary conditions from the input to the proxy using the bijective projection map, (3) solve the PDE on the proxy (which is a standard mesh), and (4) transfer back the solution on the input surface (Figure 22). Our algorithm can be directly used to solve volumetric PDEs. by calling an existing tetrahedral meshing algorithm between steps 2 and 3 (figures 1, 23). In this case, we control the geometric error by setting the target thickness (we use 2% of the longest edge of the bounding box).

Boolean Operations. The mesh arrangements algorithm enables the robust and exact (up to a final floating-point rounding) computation of Boolean operations on PWN meshes [Zhou et al. 2016]. However, the produced meshes tend to have low triangle quality that might hinder the performance of downstream algorithms. By interleaving a remeshing step performed with our algorithm after every operation, we ensure high final quality and more stable runtime. The composition of the bijections enables us to transfer properties between different nodes of the CSG tree (Figure 24).

Displacement Mapping. The middle surface of the shell is a coarse triangular mesh that can be directly used to compress the geometry of the input mesh, storing only the coarse mesh connectivity and adding the details using normal and displacement maps (Figure 25). A common way to build such displacement is to project [Collins and Hilton 2002; Kobbelt et al. 1998] the dense mesh on the coarser version. As shown in Appendix D, our method guarantees that this natural projection is also bijective as long as the coarse mesh is a section. This alleviates the loss of information even on challenging geometry configurations, and our shell can thus be used to automate the creation of projection cages and displacement maps.

Geometric Textures. The inverse projection operator provides a 2.5D parametrization around a given mesh and can be used to apply a volumetric texture (Figure 26). Note that we build the volumetric

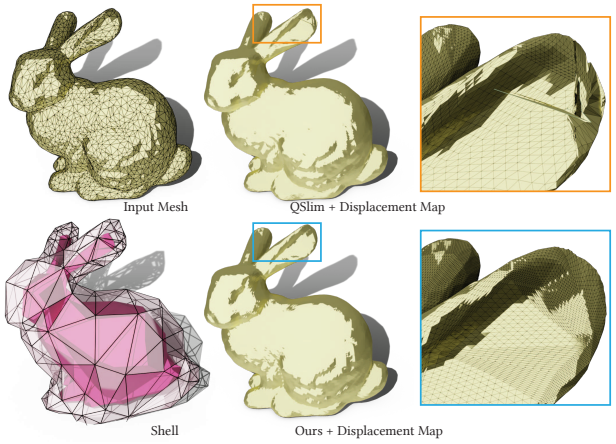


Fig. 25. Top: an input mesh decimated to create a coarse base mesh, and the details are encoded in a displacement map (along the normal) with correspondences computed with Phong projection [Kobbelt et al. 1998]. Bottom: the decimation is done within our shell while using the same projection as above. With our construction, this projection becomes bijective (Appendix D), avoiding the artifacts visible on the ears of the bunny in the top row.

Fig. 26. Two volumetric chainmail textures (right) are applied to a shell (bottom left) constructed from the *Animal mesh (top left). The original UV coordinates are transferred using our projection operator.*

texture on the simplified shell, while still being able to bijectively transfer the texture coordinates.

6 VARIANTS

Input with Self-Intersections. Up to this point, we assumed that our input meshes are without self-intersections. This requirement is necessary to guarantee a bijection between any section (e.g., the input mesh) and the middle surface. Such bijection is essential for a key target application, the transfer of boundary conditions for solving PDEs on meshes or mesh-bounded domains.

However, our method can be easily extended to meshes containing self-intersections, broadening the class of meshes it can be applied to, at the cost of making the resulting shell usable in fewer application scenarios: for example, if it is used for remeshing, it will likely generate a new surface that still contains self-intersections.

If $\mathcal{T}$ contains self-intersections, our algorithm can be trivially extended to generate a shell which will be *locally injective*, and the bijectivity of the mapping between sections still holds but with respect to the immersion. The only change required is to modify the invariance checks (Section 3.4): we have to replace the global intersection check with checking whether local triangles overlap

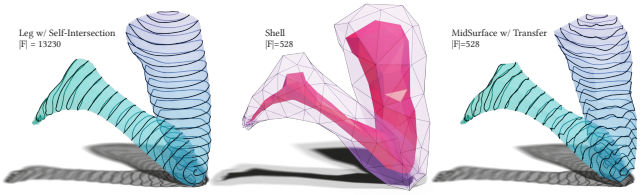


Fig. 27. An example of a shell built around a self-intersecting mesh.

Fig. 28. The *Armadillo* model with four nested cages. We create a shell from the original mesh, and then rerun our algorithm on the outer shell to create the other three layers. Note that all layers are free of self-intersections, and we have an explicit bijective map between them.

with the current prisms. Figure 27 shows an example of a mesh $\mathcal{T}$ with self-intersections, the generated shell, and the isolines of geodesic transferred on the coarser middle surface.

Resolving Shell Self-Intersections. For certain applications it might be preferable to have a shell whose top and bottom surfaces do not self-intersect: for example, in the construction of nested cages [Sacht et al. 2015] (useful for collision proxies and animation cages), we want to iteratively build nested shells while ensuring no intersections between them (Figure 28). With a small modification, our algorithm can be used to generate nested cage automatically and robustly, with the additional advantage of being able to map any quantity bijectively across the layers and to the input mesh. In contrast, [Sacht et al. 2015] does not provide guarantees on the success (e.g., the reference implementation of [Sacht et al. 2015] fails on Figure 19, probably due to the presence of a singularity).

To resolve the self-intersections of the top (bottom) surface, we identify the regions covered by more than one prism by explicitly testing intersections between the tetrahedralized prisms, accelerated using [Zomorodian and Edelsbrunner 2000]. For every detected prism, we reduce the thickness by 20%, and iterate until no more intersections are found. Differently from the procedure in Section 3.3, where reducing the thickness of the shell always maintains the validity of the shell, at this stage, the shrinking of the shell may make the shell invalid, since $\mathcal{T}$ may not be contained anymore in $\mathcal{S}$. Whenever this happens, we perform one step of red-green refinement [Bank et al. 1983] on the regions we wish to thin, and we iterate until we succeed. This procedure is guaranteed to terminate since, on the limit of the refinement, the middle surface will be geometrically identical to $\mathcal{T}$, and thus Theorem 3.5 holds. In the worst case, the procedure terminates when the size of triangles on the middle surface is comparable to the input; then no refinement is required to shrink below the minimum separation of the input.

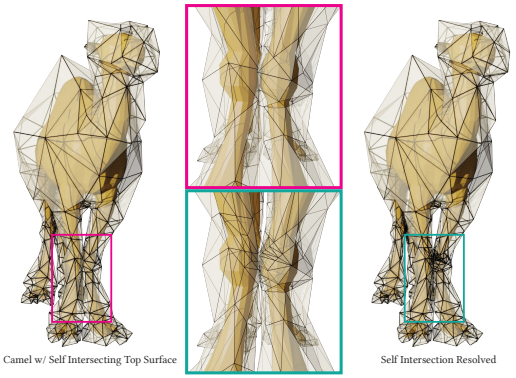


Fig. 29. After optimization, the shell may self-intersect (left). Our postprocessing can be used to extract a non-selfintersecting shell, which is easier to use in downstream applications (right).

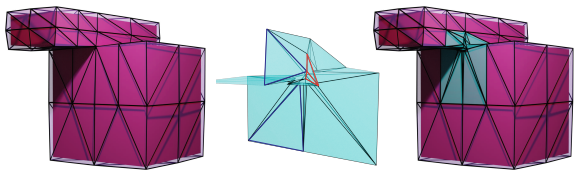


Fig. 30. We generate a pinched shell for a model with a singularity (left). Optionally, we can complete the shell using our Boolean construction.

Figure 29 shows how the intersecting shell between the legs of the camel can be shrunk to generate an intersection-free shell.

Pinching Alternative. For certain applications, such as boundary layer meshing, it is necessary to have a shell with non-zero thickness everywhere, including at singularities, and it is tolerable to lose bijectivity at the vicinity of a singularity. For these cases, we propose a Boolean construction to *fill* the shell around singularities, and to extend the projection operator $\mathcal{P}$ inside these regions. That is, every point in the filled region will project to the singularity.

Without loss of generality, let us assume that $\mathcal{T}$ has a single singularity (Figure 30). We initially construct a pinched shell, with zero thickness at the singularity, construct a valid shell (Section 3), and then perform a corefinement [Loriot et al. 2020] between a tetrahedron (centered at the singularity and whose size is smaller than the minimal thickness of the neighboring vertices) and the shell. The result of the corefinement operation (Figure 30 middle) consists of triangles belong to the tetrahedron, or the shell surface. The remaining part of the tetrahedron is a star-shaped polyhedron with the singularity in its kernel, and sharing a part of its boundary with the shell. This polyhedron can be easily tetrahedralized by connecting its triangulated boundary faces (one of them is highlighted in red in Figure 30 middle) with the singularity. For every point p in these tetrahedra, the projection operator $\mathcal{P}$ projects p to the singularity. The remaining triangles are divided into two groups: the triangles with only one new vertex complete the degenerate prisms (one of them is highlighted in blue in Figure 30 middle), while the others map to the edges they are attached to.

7 LIMITATIONS

Currently, our algorithm is limited to manifold and orientable surfaces: its extension to non-manifold and/or non-orientable meshes is a potential venue for future work. With such an extension, the integration of shells with robust tetrahedral meshers [Hu et al. 2019b, 2018] would allow to solve PDEs on imperfect triangle meshes without ever exposing the user to the volumetric mesh, allowing them to directly work on the boundary representation to specify boundary conditions and to analyze the solution of the desired PDE. Since we rely on the additional checks for the bijective constraints, our method is slower than classical surface mesh adaptation algorithms and it is not suitable for interactive applications.

Integrating our approach into existing mesh processing algorithms might lose some of their guarantees or properties since our shell might prevent some local operations. Other surface processing algorithms guarantee some properties under some regularity assumptions on the input, which might not hold when our bijective constraints are used. For example, QSlim [Garland and Heckbert 1997] might not be able to reach the desired target number of vertices and [Dey and Ray 2010] might not be able to achieve the bounded aspect ratio. A practical limitation is that integrating our approach into existing remeshing or simplification implementations requires code level access.

Our shell is ideal for triangle remeshing algorithms employing incremental changes: not every geometry processing algorithm requiring a bijective map can use our construction. For example, it is unclear how isosurface-extraction methods [Hass and Trnkova 2020] could use our shell or how global parametrization algorithms [Alliez et al. 2003; Bommes et al. 2013; Kraevoy and Sheffer 2004; Schreiner et al. 2004] could benefit from our method since they already compute a map to a common domain.

8 CONCLUDING REMARKS

We introduce an algorithm to construct shells around triangular meshes and define bijections between surfaces inside the shell. We proposed a robust algorithm to compute the shell, validated it on a large collection of models, and demonstrated its practical applicability in common applications in graphics and geometry processing.

We believe that many applications in geometry processing could benefit from bijectively mapping spatially close surfaces, and that the idea of using an explicit mesh as a common parametrization domain could be extended to the more general case of computing cross-parametrizations between arbitrary surfaces. To foster research in this direction, we will release our reference implementation as an open-source project.

ACKNOWLEDGMENTS

The authors thank all the reviewers for their constructive feedback. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This work was partially supported by the NSF CAREER award 1652515, the NSF grants IIS-1320635, DMS-1436591, DMS-1821334, OAC-1835712, OIA-1937043, CHS-1908767, CHS-1901091, a gift from Adobe Research, a gift from nTopology, and a gift from Advanced Micro Devices, Inc.

REFERENCES

Noam Aigerman, Shahar Z. Kovalsky, and Yaron Lipman. 2017. Spherical Orbifold Tutte Embeddings. *ACM Trans. Graph.* 36, 4, Article Article 90 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073615

Noam Aigerman and Yaron Lipman. 2015. Orbifold Tutte Embeddings. *ACM Trans. Graph.* 34, 6, Article Article 190 (Oct. 2015), 12 pages. https://doi.org/10.1145/2816795.2818099

Noam Aigerman and Yaron Lipman. 2016. Hyperbolic Orbifold Tutte Embeddings. *ACM Trans. Graph.* 35, 6, Article Article 217 (Nov. 2016), 14 pages. https://doi.org/10.1145/2980179.2982412

Noam Aigerman, Roi Poranne, and Yaron Lipman. 2014. Lifted Bijections for Low Distortion Surface Mappings. *ACM Trans. Graph.* 33, 4, Article Article 69 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601158

Noam Aigerman, Roi Poranne, and Yaron Lipman. 2015. Seamless Surface Mappings. *ACM Trans. Graph.* 34, 4, Article Article 72 (July 2015), 13 pages.

Pierre Alliez, Eric Colin De Verdire, Olivier Devillers, and Martin Isenburg. 2003. Isotropic surface remeshing. In *2003 Shape Modeling International.* IEEE, 49–58.

R Aubry, S Dey, EL Mestreau, and BK Karamete. 2017. Boundary layer mesh generation on arbitrary geometries. *Internat. J. Numer. Methods Engrg.* 112, 2 (2017), 157–173.

Romain Aubry and Rainald Löhner. 2008. On the 'most normal'normal. *Communications in Numerical Methods in Engineering* 24, 12 (2008), 1641–1652.

R Aubry, EL Mestreau, S Dey, BK Karamete, and D Gayman. 2015. On the 'most normal'normal—Part 2. *Finite Elements in Analysis and Design* 97 (2015), 54–63.

Chandrajit L Bajaj, Guoliang Xu, Robert J Holt, and Arun N Netravali. 2002. Hierarchical multiresolution reconstruction of shell surfaces. *Computer Aided Geometric Design* 19, 2 (2002), 89–112.

Randolph E Bank, Andrew H Sherman, and Alan Weiser. 1983. Some refinement algorithms and data structures for regular local mesh refinement. *Scientific Computing, Applications of Mathematics and Computing to the Physical Sciences* 1 (1983), 3–17.

Robert E. Barnhill, Karsten Opitz, and Helmut Pottmann. 1992. Fat surfaces: a trivariate approach to triangle-based interpolation on surfaces. *Computer Aided Geometric Design* 9, 5 (1992), 365–378. https://doi.org/10.1016/0167-8396(92)90030-s

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 51–76.

Mario Botsch and Leif Kobbelt. 2003. Multiresolution surface representation based on displacement volumes. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 483–491.

Mario Botsch, Mark Pauly, Markus H Gross, and Leif Kobbelt. 2006. PriMo: coupled prisms for intuitive surface modeling. In *Symposium on Geometry Processing*. 11–20.

Tamy Boubekeur and Marc Alexa. 2008. Phong tessellation. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 141.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding Proxies for Shape Approximation. *ACM Trans. Graph.* 36, 4, Article Article 57 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073714

Marcel Campen, Cláudio T. Silva, and Denis Zorin. 2016. Bijective Maps from Simplicial Foliations. *ACM Trans. Graph.* 35, 4, Article 74 (July 2016), 15 pages.

Frédéric Chazal and David Cohen-Steiner. 2005. A condition for isotopic approximation. *Graphical Models* 67, 5 (2005), 390–404.

Frédéric Chazal, André Lieutier, Jarek Rossignac, and Brian Whited. 2010. Ball-map: Homeomorphism between compatible surfaces. *International Journal of Computational Geometry & Applications* 20, 03 (2010), 285–306.

Yanyun Chen, Xin Tong, Jiaping Wang, Jiaping Wang, Stephen Lin, Baining Guo, Heung-Yeung Shum, and Heung-Yeung Shum. 2004. Shell texture functions. In *ACM Transactions on Graphics (TOG)*, Vol. 23. ACM, 343–353.

Xiao-Xiang Cheng, Xiao-Ming Fu, Chi Zhang, and Shuangming Chai. 2019. Practical error-bounded remeshing by adaptive refinement. *Computers & Graphics* 82 (2019), 163 – 173. https://doi.org/10.1016/j.cag.2019.05.019

Kazem Cheshmi, Danny M. Kaufman, Shoaib Kamil, and Maryam Mehri Dehnavi. 2020. NASOQ: Numerically Accurate Sparsity-Oriented QP Solver. *ACM Transactions on Graphics* 39, 4 (July 2020).

Philippe G Ciarlet. 1991. Basic error estimates for elliptic problems. *Finite Element Methods (Part 1)* (1991).

Jonathan Cohen, Dinesh Manocha, and Marc Olano. 1997. Simplifying polygonal models using successive mappings. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*. IEEE, 395–402.

Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. 1996. Simplification envelopes. In *Siggraph*, Vol. 96. 119–128.

Gordon Collins and Adrian Hilton. 2002. Mesh Decimation for Displacement Mapping.. In *Eurographics (Short Papers)*.

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

John H Conway, Heidi Burgiel, and Chaim Goodman-Strauss. 2016. *The symmetries of things*. CRC Press.

Harold Scott Macdonald Coxeter. 1973. *Regular polytopes*. Courier Corporation.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)* 32, 5 (2013).

Tamal K Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V Nekhayev. 1999. Topology preserving edge contraction. *Publ. Inst. Math.(Beograd)(NS)* 66, 80 (1999), 23–45.

Tamal K Dey and Tathagata Ray. 2010. Polygonal surface remeshing with Delaunay refinement. *Engineering with computers* 26, 3 (2010), 289–301.

Julien Dompierre, Paul Labbé, Marie-Gabrielle Vallet, and Ricardo Camarero. 1999. How to Subdivide Pyramids, Prisms, and Hexahedra into Tetrahedra. *IMR* 99 (1999), 195.

Marion Dunyach, David Vanderhaeghe, Loïc Barthe, and Mario Botsch. 2013. Adaptive remeshing for real-time mesh deformation.

Ramsay Dyer, Hao Zhang, and Torsten Möller. 2007. Delaunay mesh construction. (2007).

Hans-Christian Ebke, Marcel Campen, David Bommes, and Leif Kobbelt. 2014. Level-of-detail quad meshing. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 184.

Danielle Ezuz, Justin Solomon, and Mirela Ben-Chen. 2019. Reversible Harmonic Maps between Discrete Surfaces. *ACM Trans. Graph.* 38, 2, Article Article 15 (March 2019), 12 pages. https://doi.org/10.1145/3202660

Michael Floater. 2003. One-to-one piecewise linear mappings over triangulations. *Math. Comp.* 72, 242 (2003), 685–696.

Michael S. Floater. 1997. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design* 14 (1997), 231–250.

Michael S. Floater and Kai Hormann. 2005. Surface Parameterization: a Tutorial and Survey. In *In Advances in Multiresolution for Geometric Modelling, Mathematics and Visualization*. Springer Verlag, 157–186.

Xiao-Ming Fu and Yang Liu. 2016. Computing Inversion-free Mappings by Simplex Assembly. *ACM Trans. Graph.* 35, 6, Article 216 (Nov. 2016), 12 pages.

Rao V Garimella and Mark S Shephard. 2000. Boundary layer mesh generation for viscous flow simulations. *Internat. J. Numer. Methods Engrg.* 49, 1-2 (2000), 193–218.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 209–216.

Michael Garland and Paul S Heckbert. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*. IEEE, 263–269.

Bernd Gärtner and Sven Schönherr. 2000. An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proceedings of the sixteenth annual symposium on Computational geometry*. 110–118.

Craig Gotsman and Vitaly Surazhsky. 2001. Guaranteed intersection-free polygon morphing. *Computers & Graphics* 25, 1 (2001), 67–75.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3.

André Guéziec. 1996. *Surface simplification inside a tolerance volume*. IBM TJ Watson Research Center.

Philippe Guigue and Olivier Devillers. 2003. Fast and robust triangle-triangle overlap test using orientation predicates. *Journal of graphics tools* 8, 1 (2003), 25–32.

George W Hart. 2018. Conway notation for polyhedra. *URL: http://www. gergehart. co/virtual-polyhedra/conway_notation. html* (2018).

J Hass and M Trnkova. 2020. Approximating isosurfaces by guaranteed-quality triangular meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 29–40.

Kai Hormann and Günther Greiner. 2000. *MIPS: An efficient global parametrization method*. Technical Report. ERLANGEN-NUERNBERG UNIV (GERMANY) COMPUTER GRAPHICS GROUP.

Kai Hormann, Bruno Lévy, and Alla Sheffer. 2007. Mesh Parameterization: Theory and Practice. In *ACM SIGGRAPH 2007 Courses* (San Diego, California) *(SIGGRAPH '07)*. ACM, New York, NY, USA.

K. Hormann and N. Sukumar (Eds.). 2017. *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics*. CRC Press, Boca Raton, FL.

K. Hu, D. Yan, D. Bommes, P. Alliez, and B. Benes. 2017. Error-Bounded and Feature Preserving Surface Remeshing with Minimal Angle Improvement. *IEEE Transactions on Visualization and Computer Graphics* 23, 12 (Dec 2017), 2560–2573. https://doi.org/10.1109/TVCG.2016.2632720

Kaimo Hu, Dong-Ming Yan, David Bommes, Pierre Alliez, and Bedrich Benes. 2016. Error-bounded and feature preserving surface remeshing with minimal angle improvement. *IEEE transactions on visualization and computer graphics* 23, 12 (2016), 2560–2573.

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019a. TriWild: robust triangulation with curve constraints. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 52.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2019b. Fast Tetrahedral Meshing in the Wild. *arXiv preprint arXiv:1908.03581* (2019).

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), 60–1.

Jin Huang, Xinguo Liu, Haiyang Jiang, Qing Wang, and Hujun Bao. 2007. Gradient-based shell generation and deformation. *Computer Animation and Virtual Worlds*

18, 4-5 (2007), 301–309.
Alec Jacobson, Daniele Panozzo, C Schüller, O Diamanti, Q Zhou, N Pietroni, et al. 2016. libigl: A simple C++ geometry processing library, 2016.
Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial complex augmentation framework for bijective maps. *ACM Transactions on Graphics* 36, 6 (2017).
Xiangmin Jiao and Michael T Heath. 2004. Overlaying surface meshes, part I: Algorithms. *International Journal of Computational Geometry & Applications* 14, 06 (2004), 379–402.
M. Jin, J. Kim, F. Luo, and X. Gu. 2008. Discrete Surface Ricci Flow. *IEEE Transactions on Visualization and Computer Graphics* 14, 5 (Sep. 2008), 1030–1043. https://doi.org/10.1109/TVCG.2008.57
Yao Jin, Dan Song, Tongtong Wang, Jin Huang, Ying Song, and Lili He. 2019. A shell space constrained approach for curve design on surface meshes. *Computer-Aided Design* 113 (2019), 24–34.
Liliya Kharevych, Boris Springborn, and Peter Schröder. 2006. Discrete Conformal Mappings via Circle Patterns. *ACM Trans. Graph.* 25, 2 (April 2006), 412–438. https://doi.org/10.1145/1138450.1138461
Peter Knabner and Gerhard Summ. 2001. The invertibility of the isoparametric mapping for pyramidal and prismatic finite elements. *Numer. Math.* 88, 4 (2001), 661–681.
Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Siggraph*, Vol. 98. 105–114.
Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Vladislav Kraevoy and Alla Sheffer. 2004. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 861–869.
Aaron Lee, Henry Moreton, and Hugues Hoppe. 2000. Displaced subdivision surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 85–94.
Aaron WF Lee, Wim Sweldens, Peter Schröder, Lawrence C Cowsar, and David P Dobkin. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. In *Siggraph*, Vol. 98. 95–104.
Jerome Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. 2001. Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM, 227–232.
Bruno Lévy. 2015. Geogram.
Minchen Li, Danny M Kaufman, Vladimir G Kim, Justin Solomon, and Alla Sheffer. 2018. OptCuts: joint optimization of surface cuts and parameterization. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–13.
Yaron Lipman. 2014. Bijective mappings of meshes with boundary and the degree in mesh processing. *SIAM Journal on Imaging Sciences* 7, 2 (2014), 1263–1283.
Nathan Litke, Marc Droske, Martin Rumpf, and Peter Schröder. 2005. An image processing approach to surface matching.. In *Symposium on Geometry Processing*, Vol. 255.
Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam I Gingold. 2017. Seamless: seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.* 36, 6 (2017).
Yong-Jin Liu, Chun-Xu Xu, Dian Fan, and Ying He. 2015. Efficient construction and simplification of Delaunay meshes. *ACM Transactions on Graphics (TOG)* 34, 6 (2015).
Sébastien Loriot, Mael Rouxel-Labbé, Jane Tournois, and Ilker O. Yaz. 2020. Polygon Mesh Processing. In *CGAL User and Reference Manual* (5.0.3 ed.). CGAL Editorial Board. https://doc.cgal.org/5.0.3/Manual/packages.html#PkgPolygonMeshProcessing
Manish Mandad, David Cohen-Steiner, and Pierre Alliez. 2015. Isotopic approximation within a tolerance volume. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 64.
Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph.* 36, 4 (2017), 71–1.
Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.
Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2015. Air Meshes for Robust Collision Handling. *ACM Trans. Graph.* 34, 4 (July 2015), 9.
Hubert Nguyen. 2007. *Gpu gems 3*. Addison-Wesley Professional.
Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. 2012. Functional Maps: A Flexible Representation of Maps between Shapes. *ACM Trans. Graph.* 31, 4, Article Article 30 (July 2012), 11 pages. https://doi.org/10.1145/2185520.2185526
Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodolà, Mirela Ben-Chen, Leonidas Guibas, Frederic Chazal, and Alex Bronstein. 2017. Computing and Processing Correspondences with Functional Maps. In *ACM SIGGRAPH 2017 Courses (SIGGRAPH '17)*. Article Article 5. https://doi.org/10.1145/3084873.3084877
Daniele Panozzo, Ilya Baran, Olga Diamanti, and Olga Sorkine-Hornung. 2013. Weighted averages on surfaces. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 60.
Jianbo Peng, Daniel Kristjansson, and Denis Zorin. 2004. Interactive modeling of topologically complex geometric detail. In *ACM Transactions on Graphics (TOG)*, Vol. 23. ACM, 635–643.
Bui Tuong Phong. 1975. Illumination for computer generated pictures. *Commun. ACM* 18, 6 (1975), 311–317.
Nico Pietroni, Marco Tarini, and Paolo Cignoni. 2010. Almost Isometric Mesh Parameterization through Abstract Domains. *IEEE Transactions on Visualization and Computer Graphics* 16, 4 (July 2010), 621–635. https://doi.org/10.1109/TVCG.2009.96
Serban D Porumbescu, Brian Budge, Louis Feng, and Kenneth I Joy. 2005. Shell maps. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM, 626–633.
Emil Praun, Wim Sweldens, and Peter Schröder. 2001. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 179–184.
Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2, Article 16 (2017), 16 pages.
Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 170.
Patrick Schmidt, Janis Born, Marcel Campen, and Leif Kobbelt. 2019. Distortion-minimizing injective maps between surfaces. *ACM Transactions on Graphics (TOG)* 38 (2019).
John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. 2004. Inter-Surface Mapping. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 870–877. https://doi.org/10.1145/1015706.1015812
Christian Schüller, Ladislav Kavan, Daniele Panozzo, and Olga Sorkine-Hornung. 2013. Locally Injective Mappings. In *Symposium on Geometry Processing*. 125–135.
Nicholas Sharp and Keenan Crane. 2020. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)* 39, 5 (2020).
Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019. Navigating intrinsic triangulations. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 55.
Alla Sheffer, Emil Praun, and Kenneth Rose. 2006. Mesh Parameterization Methods and Their Applications. *Found. Trends. Comput. Graph. Vis.* 2, 2 (2006), 105–171.
Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18, 3 (1997).
Meged Shoham, Amir Vaxman, and Mirela Ben-Chen. 2019. Hierarchical Functional Maps between Subdivision Surfaces. *Computer Graphics Forum* 38, 5 (2019), 55–73. https://doi.org/10.1111/cgf.13789
Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (July 2015), 9 pages.
Boris Springborn, Peter Schröder, and Ulrich Pinkall. 2008. Conformal Equivalence of Triangle Meshes. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–11. https://doi.org/10.1145/1360612.1360676
B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. 2017. OSQP: An Operator Splitting Solver for Quadratic Programs. *ArXiv e-prints* (Nov. 2017). arXiv:math.OC/1711.08013
Kenneth Stephenson. 2005. *Introduction to circle packing: The theory of discrete analytic functions*. Cambridge University Press.
Vitaly Surazhsky and Craig Gotsman. 2001. Morphing stick figures using optimized compatible triangulations. In *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on*. IEEE, 40–49.
The CGAL Project. 2020. *CGAL User and Reference Manual* (5.0.3 ed.). CGAL Editorial Board. https://doc.cgal.org/5.0.3/Manual/packages.html
Jean-Marc Thiery, Julien Tierny, and Tamy Boubekeur. 2012. CageR: Cage-Based Reverse Engineering of Animated 3D Shapes. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2303–2316. https://doi.org/10.1111/j.1467-8659.2012.03159.x
W. T. Tutte. 1963. How to draw a Graph. *Proceedings of the London Mathematical Society* 13, 3 (1963), 743–768.
Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. 2003. View-dependent displacement mapping. In *ACM Transactions on graphics (TOG)*, Vol. 22. ACM, 334–339.
Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. 2004. Generalized displacement maps. In *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques*. Eurographics Association, 227–233.
Ofir Weber and Denis Zorin. 2014. Locally Injective Parametrization with Arbitrary Fixed Boundaries. *ACM Trans. Graph.* 33, 4, Article 75 (July 2014), 12 pages.
Eugene Zhang, Konstantin Mischaikow, and Greg Turk. 2005. Feature-based Surface Parameterization and Texture Mapping. *ACM Trans. Graph.* 24, 1 (Jan. 2005), 1–27.
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–15.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).
Afra Zomorodian and Herbert Edelsbrunner. 2000. Fast software for box intersections. In *Proceedings of the sixteenth annual symposium on Computational geometry*. 129–138.

A PROOFS

A.1 Theorem 3.2

Consider the unique (up to symmetry) piece-wise affine map T deforming Δ into a reference prism $\hat{\Delta} = \{u, v \geq 0 | u+v \leq 1\} \times [-1, 1]$, identifying the bottom surface of $\hat{\Delta}$ as $z = -1$, middle surface as $z = 0$, and top surface as $z = 1$. Let $T_{\boldsymbol{T}}$ be the affine map mapping tetrahedron $\boldsymbol{T}$ to the reference prism. Since the volume of all $\boldsymbol{T}$ is positive by assumption, the map T is bijective and orientation preserving. In particular, T transforms $\mathcal{V}(p)$ to the const vector $e_z = (0, 0, 1)$ since the edges of the prism are mapped to axis aligned edges of the reference prism. The projection operator $\mathcal{P}(p)$ can thus be equivalently defined as $\mathcal{P}(p) = T^{-1}(\mathcal{P}_z(T(p)))$, where $\mathcal{P}_z$ is the projection over the z-axis in the reference prism. $\mathcal{P}_z$ is bijective if the piecewise linear mesh intersecting Δ is composed of triangles with positive area (and the boundaries are mapped to the boundaries) [Lipman 2014], after being mapped to the reference Δ and projected by $\mathcal{P}_z$. In the reference domain, having positive area after projection (with a fixed boundary) is equivalent to that the dot product between the projection direction and the normal of every face is positive. In the top half of Δ, $\mathcal{P}_z$ is bijective if for every point $T(p), p \in f$ $T(n_p) \cdot [0, 0, 1] = T(n_p) \cdot T(\mathcal{V}(p)) > 0$ which holds since $n_p \cdot \mathcal{V}(p) > 0$ (from the definition of section) and the fact that T is orientation preserving and thus not changing the sign of the dot product.

A.2 Proposition 3.4

For simplicity, we will show that a uniform lower bound δ for all vertices exists. By consider a triangle of T, extruded over the displacement field $\mathcal{N}$, we obtain a generalized half prism with six vertices $0, e_1, e_2, \delta n_0, e_1 + \delta n_1, e_2 + \delta\, n_2$, with δ a positive constant. The indices of the tetrahedra are:

(0 ,2, 3, 4), (0 ,3, 4, 5), (0 ,1, 5, 3), (0 ,1, 2, 3), (1 ,2, 3, 4), (0 ,1, 2, 4), (0 ,1, 5, 4), (1 ,3, 4, 5), (1 ,2, 3, 5), (0 ,2, 5, 4), (0 ,1, 2, 5), (2 ,3, 4, 5).

For a sufficiently small δ, the linear term will dominate the higher power of δ, which we omit. The dominant volume terms are listed in the following table:

vol1$(0, e_2, \delta n_0, e_1 + \delta n_1)$	$\delta\langle e_2, n_0, e_1\rangle$
vol2$(0, \delta n_0, e_1 + \delta n_1, e_2 + \delta n_2)$	$\delta\langle n_0, e_1, e_2\rangle$
vol3$(0, e_1, e_2 + \delta n_2, \delta n_0)$	$\delta\,\langle e_1, e_2, n_0\rangle$
vol4$(0, e_1, e_2, \delta n_0)$	$\delta\langle e_1, e_2, n_0\rangle$
vol5$(e_1, e_2, \delta n_0, e_1 + \delta n_1)$	$\langle e_2 - e_1, \delta n_0 - e_1, \delta n_1\rangle$ $= \delta\langle e_2, -e_1, n_1\rangle$
vol6$(0, e_1, e_2, e_1 + \delta n_1)$	$\delta\langle e_1, e_2, n_1\rangle$
vol7$(0, e_1, e_2 + \delta n_2, e_1 + \delta n_1)$	$\langle e_1, e_2 + \delta n_2, \delta n_1\rangle$ $= \delta\langle e_1, e_2, n_1\rangle$
vol8$(e_1, \delta n_0, e_1 + \delta n_1, e_2 + \delta n_2)$	$\delta\langle \delta n_0 - e_1, n_1, e_2 - e_1 + \delta n_2\rangle$ $= \delta\langle -e_1, n_1, e_2\rangle$
vol9$(e_1, e_2, \delta n_0, e_2 + \delta n_2)$	$\langle e_2 - e_1, \delta n_0 - e_1, e_2 + \delta n_2 - e_1\rangle$ $= \delta\langle e_2 - e_1, \delta n_0 - e_1, n_2\rangle$ $= \delta\langle e_2, -e_1, n_2\rangle$
vol10$(0, e_2, e_2 + \delta n_2, e_1 + \delta n_1)$	$\delta\langle e_2, n_2, e_1\rangle$
vol11$(0, e_1, e_2, e_2 + \delta n_2)$	$\delta\langle e_1, e_2, n_2\rangle$
vol12$(e_2, \delta n_0, e_1 + \delta n_1, e_2 + \delta n_2)$	$\langle \delta n_0 - e_2, e_1 + \delta n_1 - e_2, \delta n_2\rangle$ $= \delta\langle -e_2, e_1, n_2\rangle$

For all the 12 tetrahedra the linear term is multiplied by a determinant containing the edges of the prism. We check directly that all these determinants are positive due to the assumption that $\mathrm{C}\mathcal{N}_i > 0$.

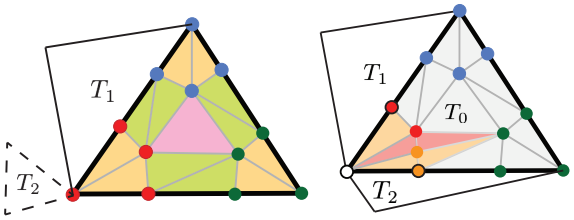


Fig. 31. Bevel patterns used in the proofs in Appendix A.4 and Appendix B

A.3 Theorem 3.5

Consider a ball around a point p of the middle face M of the prism. If the radius $r(p) > 0$ is sufficiently small, it contains, at most, a single vertex of M, and parts of incident faces. As M is compact, there is a minimal value of r on M, and it is positive. The line along the normal, passing through v, intersects incident faces at v so it cannot intersect them at any other point. Thus, the top and bottom prism vertices can be obtained by displacing V by r in either direction along the normal. Consider the r-neighborhood of the M, i.e. the union of all balls of radius r centered at points of M. This is a convex set, containing only M and parts of vertex-adjacent faces. All top and bottom prism vertices are in this set. Thus tetrahedra connecting these vertices are also in the set, i.e., complete prisms. The fact that tetrahedra are nondegenerate is established in Proposition 3.4.

A.4 Theorem 3.6

To show that $\mathcal{T}$ is a section of $\mathcal{S}$ we need to show that (1) $M_\Delta = \mathcal{T} \cap \Delta$ is a simply connected patch for any prism of $\mathcal{S}$ and (2) for each prism Δ and every point $p \in M_\Delta$ the dot product between the face normal $n(p)$ and the pillars $\mathcal{V}_i, i = 1, 2, 3$ is strictly positive.

(1) is trivial since for every prism Δ of $\mathcal{S}$, Δ contains exactly one triangle of the topological bevel refinement face of $\mathcal{T}$.

To prove (2), consider the dot products of the normals of each of the triangles in the beveled region sharing at least a vertex with M_Δ, and the vectors along the pillars of M_Δ. We use colors to refer to the triangles of beveled prisms as shown in Figure) 31 left.

The prism corresponding to the pink triangle covers only the interior of the original triangle. It satisfies the dot product condition since its pillars are copied from the solution of Problem (3) at each vertex. A similar argument can be made for the green prisms: each prism gets its pillars from the edges incident at two of the vertices, which are compatible with the normal of the adjacent triangle (e.g., red and blue pillars have positive dot product with the normal of T_1). Finally, the prisms corresponding to orange triangles cover the same one ring that was used to compute the original pillars (e.g., red pillar is compatible with T_1 and T_2). Since the same pillar is used for each of the 3 vertices of the original prism, the dot product is unchanged.

A.5 Theorem 3.7

We use A° to denote the interior of a set A. We first assert that the topological disk patch $\mathcal{T} \cap C$ coincides with $\mathcal{T} \cap C'$. To make the notation more concise, we define $\mathcal{T}_C = \mathcal{T} \cap C$ and $\mathcal{T}_{C'} = \mathcal{T} \cap C'$.

We show that $\mathcal{T}_C \subset C'$ by contradiction: assume there is $p \in \mathcal{T}_C$ but $p \notin C'$. Choose a point $q \in \partial\mathcal{T}_C$, that belongs to single prisms

in C and C' and denote the enclosing prism in C as Δ_q and the corresponding one in C' as Δ'_q. It follows from the positivity of the volumes of Δ_q and Δ'_q (Assumption 1), that there is $\varepsilon > 0$ such that the half-ball (for sufficiently small ε) $\text{Ball}(q, \varepsilon) \cap \Delta_q$ coincides with $\text{Ball}(q, \varepsilon) \cap \Delta'_q$. Futhermore, the validity of the initial shell guarantees that there exists an interior point $\bar{q} \in \text{Ball}(q, \varepsilon) \cap \Delta_q^\circ \cap \mathcal{T}$, and $\bar{q} \in \Delta_q'^\circ \cap \mathcal{T} \subset C'^\circ \cap \mathcal{T}$.

Since neither p or $\bar{q}$ lies on the boundary side triangles of $\partial C'$ (since $p, \bar{q} \notin \partial\mathcal{T}_C$), therefore there exists an interior path P (i.e., a path both in $\mathcal{T}_C$ and in $\mathcal{T}_{C'}$) that connects p and $\bar{q}$. By continuity the path P it will cross $\partial C'$ at some point $p_i \in \mathcal{T}_C$. Since the $\partial\mathcal{T}_{C'} = \partial\mathcal{T}_C$ is a fixed boundary loop (by the definition of $\mathbb{O}$), and does not contain interior points, p_i can only cross on the top surface or bottom surface of C' which contradicts Assumption 2. The roles of C and C' can be inverted to complete the proof.

We then map M_C to a regular planar n-gon D, with vertices on ∂M_C mapped cyclically to the vertices of ∂D. It follows from [Floater 2003, Corollary 6.2] that such a convex combination mapping $\varphi\colon M_C \to D$ is bijective. Then, similarly to the construction in Theorem 3.2, we define $\phi\colon C \to D \times [-1, 1]$ through affine mapping induced by the thetrahedralization of the prisms Δ_i. Similarly, we define $\psi\colon C' \to D \times [-1, 1]$. The mapping ψ is also bijective because of Assumption 1 and the definition of $\mathbb{O}$ it follows that the codomain of ψ is the same as ϕ. Note that in both cases, the map transforms the vector field $\mathcal{V}(p)$ to the constant vector field $e_z = (0, 0, 2)$, and the projection is $\mathcal{P}_z$, as defined in Appendix A.1.

The dot product condition (Assumption 3) ensures that $\mathcal{P}_z$ defines a bijection between D and $\psi(\mathcal{T}_C)$; and further, the image satisfies $\mathcal{P}_z(\psi(\Delta'_i \cap \mathcal{T})) = \psi(M'_i)$ where M'_i is the middle surface (a single triangle) of Δ'_i. Therefore, since ψ and $\mathcal{P}_z$ are both continuous and bijective, they are homeomorphisms which preserve topology, thus $\Delta'_i \cap \mathcal{T}$ is a simply connected topological disk.

B EXTENSION TO MESHES WITH SINGULARITIES

Most of the proofs and definition in Section 3 easily extends to meshes with singularity or pinched shells. For instance, both Theorem 3.2 and Definition 3.3 apply to pinched shells by just considering a prism made of 4 (2 for the top and 2 for the bottom slab) instead of 6. Propositions 3.4 and 3.5 both rely on per-vertex properties; by just excluding singular vertices (and neighboring prism) from the statements the proofs (and our algorithm) still holds.

Theorem 3.7 requires some minor additional considerations. As a consequence of beveling, no two singularities are adjacent. Therefore, we can always find a point $q \in \partial\mathcal{T}_C$ that belongs to a single prism (Appendix A.5) since every prism will have a positive volume because no singularities are adjacent.

Finally Proposition 3.6 requires a new proof since the beveling pattern used for singularities is different.

Proof. In Figure 31 right, we highlight in red and orange the triangles not covered by the discussion in Appendix A.4. The prism corresponding to an orange middle triangle intersects the edge-adjacent triangle. And since the new pillars are computed as the average of the normals of the two adjacent triangles with dihedral angle strictly less than 360° (Section 3.5 inset), each dot product is positive. The prism for a pink middle triangle intersects only the interior of the original triangle, and the dot product between the pillars and the triangle normal are positive by construction. □

C REDUCTION TO A STANDARD QP

With slack variable t, Problem 2 can be rewritten as

$$\max_t t;\ d_v n_f \geq t, \|d_v\| = 1, t \geq \epsilon.$$

In the admissible region, substituting $t = 1/s$,

$$\min_s s;\ s d_v n_f \geq 1, \|d_v\| = 1, 0 < s \leq 1/\epsilon.$$

Substituting $x = s d_v$ (thus $\|x\| = s\|d\| = s$), we obtain the QP problem 3. Note that the lower bound on s is implied from $s d_v n_f \geq 1$, and the upper bound $\|x\| < 1/\epsilon$ is checked a posteriori.

D BIJECTIVITY OF THE NONLINEAR PRISMATIC TRANSFORMATION

In this section, we show that the isoparametric transformation of prismatic finite element [Ciarlet 1991] is also bijective if I1 holds.

We follow the notation from Appendix A.2. The map is defined as (for the top slab)

$$f(u, v, \eta) = u e_1 + v e_2 + \eta n_0 + u\eta n_{01} + v\eta n_{02},$$
$$\det J_f = \langle (1 - u - v) n_0 + u n_1 + v n_2, e_1 + \eta n_{01}, e_2 + \eta n_{02} \rangle,$$

where η is the variable corresponding to the thickness direction, and $n_{ij} = n_j - n_i$. Observe that $\det J_f(u, v, \eta_0)$ is linear in u, v for fixed η_0, the extrema will only be achieved at the corner points [Knabner and Summ 2001]. Therefore, it is sufficient to check the signs at three edges $(0, 0, \eta)$, $(1, 0, \eta)$, $(0, 1, \eta)$ for $\eta \in [0, 1]$.

$$\begin{aligned}
&\det J_f(0, 0, \eta) \\
&= \langle n_0, (1 - \eta) e_1 + \eta(e_1 + n_1 - n_0), (1 - \eta) e_2 + \eta(e_2 + n_2 - n_0) \rangle \\
&= (1 - \eta)^2 \langle n_0, e_1, e_2 \rangle + (1 - \eta)\eta \langle n_0, e_1, e_2 + n_2 \rangle \\
&+ (1 - \eta)\eta \langle n_0, e_1 + n_1, e_2 \rangle + \eta^2 \langle n_0, e_1 + n_1, e_2 + n_2 \rangle \\
&= (1 - \eta)^2 \text{vol}_4 + (1 - \eta)\eta(\text{vol}_3 + \text{vol}_1) + \eta^2 \text{vol}_2 > 0.
\end{aligned}$$

By symmetry, it is easy to verify that the following are positive,

$$\det J_f(1, 0, \eta) = (1 - \eta)^2 \text{vol}_6 + (1 - \eta)\eta(\text{vol}_7 + \text{vol}_5) + \eta^2 \text{vol}_8,$$
$$\det J_f(0, 1, \zeta) = (1 - \eta)^2 \text{vol}_{11} + (1 - \eta)\eta(\text{vol}_9 + \text{vol}_{10}) + \eta^2 \text{vol}_{12},$$

E DOUBLE SLAB

In Section 3.1, we explain that we want to ensure that our shell validity is independent from the enumeration of the faces; for this reason, we require that the conditions I1 and I2 are satisfied for all possible decompositions. Without using the double slab, different decompositions of the prism will result in different intersections with the middle surface. In other words, the projection operator will project the input mesh to a different set of triangles on the middle surface. The double slab makes the middle surface independent of the decomposition by construction. We note that Theorem 3.6 is valid only for double-slab prisms since the statement requires that one prism contains only one triangle.