Bijective and Coarse High-Order Tetrahedral Meshes

ZHONGSHI JIANG, ZIYI ZHANG, and YIXIN HU, New York University, USA
TESEO SCHNEIDER, University of Victoria, Canada and New York University, USA
DENIS ZORIN and DANIELE PANOZZO, New York University, USA

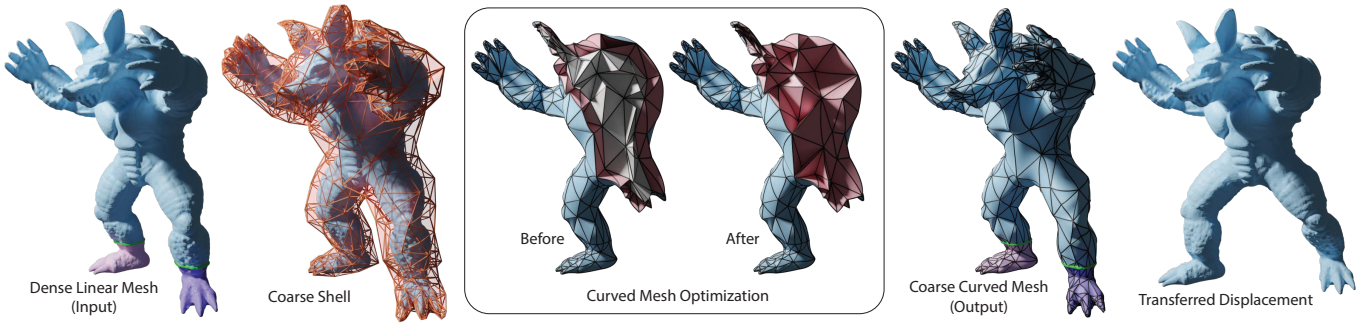


Fig. 1. Our pipeline starts from a dense *linear* mesh with annotated features (green), and converts it into a curved shell filled with high-order elements. The region bounded by the shell is then tetrahedralized with linear elements, followed by optimization. Our output is a coarse yet accurate, curved tetrahedral mesh ready to be used in the finite element method based simulation. Our construction also provides a bijective map between the input surface and the boundary of the final tetrahedral mesh, which is used to transfer attributes and boundary conditions.

We introduce a robust and automatic algorithm to convert linear triangle meshes with feature annotated into coarse tetrahedral meshes with curved elements. Our construction guarantees that the high-order meshes are free of element inversion or self-intersection. A user-specified maximal geometrical error from the input mesh controls the faithfulness of the curved approximation. The boundary of the output mesh is in bijective correspondence to the input, enabling attribute transfer between them, such as boundary conditions for simulations, making our curved mesh an ideal replacement or complement for the original input geometry.

The availability of a bijective shell around the input surface is employed to ensure robust curving, prevent self-intersections, and compute a bijective map between the linear input and curved output surface. As necessary building blocks of our algorithm, we extend the bijective shell formulation to support features and propose a robust approach for boundary-preserving linear tetrahedral meshing.

We demonstrate the robustness and effectiveness of our algorithm by generating high-order meshes for a large collection of complex 3D models.

CCS Concepts: • **Mathematics of computing** → **Mesh generation**; • **Computing methodologies** → *Shape modeling*.

Additional Key Words and Phrases: High-order mesh generation, Bijective map, Feature preserving, Mesh adaptation, Attribute transfer

Authors' addresses: Zhongshi Jiang, jiangzs@nyu.edu; Ziyi Zhang, ziyizhang@nyu.edu; Yixin Hu, yixin.hu@nyu.edu; Teseo Schneider, teseo@uvic.ca; Denis Zorin, dzorin@cs.nyu.edu; Daniele Panozzo, panozzo@nyu.edu. Courant Institute of Mathematical Sciences, New York University, 60 5th Avenue, New York, NY 10011, USA.

0730-0301/2021/8-ART157 $15.00
https://doi.org/10.1145/3450626.3459840

ACM Reference Format:
Zhongshi Jiang, Ziyi Zhang, Yixin Hu, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2021. Bijective and Coarse High-Order Tetrahedral Meshes. *ACM Trans. Graph.* 40, 4, Article 157 (August 2021), 16 pages. https://doi.org/10.1145/3450626.3459840

1 INTRODUCTION

Piecewise linear approximations of surfaces are a popular representation for 3D geometry due to their simplicity and wide availability of libraries and algorithms to process them. However, dense sampling is required to faithfully approximate smooth surfaces. Curved meshes, that is, meshes whose element's geometry is described as a high-order polynomial, are an attractive alternative for many applications, as they require fewer elements to achieve the same representation accuracy of linear meshes. In particular, curved meshes have been shown to be effective in a variety of simulation settings in mechanical engineering, computational fluid dynamics, and graphics. Despite their major benefits, they are not as popular as linear meshes: We believe that one of the main reasons among others (e.g., contact resolution, interactive manipulation, texture mapping, and distance computation) for their limited usage is the lack of an automatic, robust way of constructing them.

While robust meshing algorithm exists for volumetric linear tetrahedral meshing, there are few algorithms for curved meshes, and even fewer of them having either a commercial or open-source implementation (Section 2). Only a few algorithms work directly on arbitrary triangle meshes (most of them require the input geometry to be either a CAD file or an implicit function), and none of them can reliably process a large collection of 3D models.

We propose the first robust and automatic algorithm to convert dense piecewise linear triangle meshes (which can be extracted from scanned data, volumetric imaging, or CAD models) into coarse,

curved tetrahedral meshes equipped with a bijective map between the input triangle mesh and the boundary of the tetrahedral mesh. Our algorithm takes advantage of the recently proposed bijective shell construction [Jiang et al. 2020] to allow joint coarsening, remeshing, and curving of the dense input mesh, which we extend to support feature annotations. Our outputs are guaranteed to have a self-intersection-free boundary, and the geometric map of every element is guaranteed to be bijective, an important requirement for FEM applications. Note that our guarantees hold using exact computations. We leave the development of floating-point predicates as future work.

The key ingredient of our algorithm is the separation of the curved volumetric meshing problem into a near-surface, shell curving problem [Moxey et al. 2015; Sherwin and Peiró 2002], followed by a restricted type of linear volumetric meshing.

We believe that our curved meshing algorithm will enable wider adoption of curved meshes, as it will provide a way to automatically convert geometric data in multiple formats into a coarse tetrahedral mesh readily usable in finite element applications. To showcase the benefits of our approach, we study two settings: (1) we show that a coarse proxy mesh can be used to compute non-linear deformations efficiently and transfer them onto a high-resolution geometry, targeting real-time simulation (Figure 1), and (2) we show that our meshes are ready to use in downstream FEM simulations.

We validate the reference implementation of our approach on a large collection of more than 8000 geometrical models, which will be released as an open-source project[1] to foster the adoption of curved meshes in academia and industry.

Our contributions are:

- An algorithm to convert dense piecewise linear meshes into coarse curved volumetric meshes while preserving a bijective map with the input and bounding the approximation error.
- An extension of the bijective shell construction algorithm to support annotated features.
- An algorithm for conforming tetrahedral meshing without allowing refinement on the boundary, but allowing internal Steiner points.
- A large-scale dataset of high-order tetrahedral meshes.

2 RELATED WORKS

We review the literature on the generation of unstructured and structured curved meshes. We also review the literature on boundary-preserving linear meshing, as it is an intermediate step of our algorithm.

2.1 Curved Tetrahedral Mesh Generation

High-order meshes are used in applications in graphics [Bargteil and Cohen 2014; Mezger et al. 2009; Suwelack et al. 2013] and engineering analysis [Jameson et al. 2002] where it is important to reduce the geometric discretization error [Babuska and Guo 1988; Babuška and Guo 1992; Bassi and Rebay 1997; Luo et al. 2001; Oden 1994], while using a low number of degrees of freedom. The creation of high-order meshes is typically divided into three steps: (1) linear meshing of the smooth input surface, (2) curving of the linear elements to fit the surface, and (3) optimization to heal the elements inverted during curving. We first cover steps 2 and 3 and postpone the overview of linear tetrahedral algorithms to Section 2.3.

Direct methods. Direct methods are the simplest family of curving algorithms, as they explicitly interpolate a few points of the target curved surface or project the high-order nodes on the curved boundary [Abgrall et al. 2012; Dey et al. 1999; Ghasemi et al. 2016; Marcon et al. 2019; Moxey et al. 2015; Sherwin and Peiró 2002; Turner 2017]. The curved elements are represented using Lagrange polynomials, [Dey et al. 1999; Peiró et al. 2008], quadratic or cubic Bézier polynomials [George and Borouchaki 2012; Lu et al. 2013; Luo et al. 2002a], or NURBS [Engvall and Evans 2016, 2017]. [Shephard et al. 2005; Sherwin and Peiró 2002] further optimizes the high-order node distribution according to geometric quantities of interest, such as length, geodesic distance, and curvature. In the case where no CAD information is available, [Wang et al. 2016], [Jiao and Wang 2012] use smooth reconstruction to compute high-order nodes and perform curving.

Deformation methods. Deformation methods consider the input linear mesh as a deformable, elastic body, and use controlled forces to deform it to fit the curved boundary. Different physical models have been employed such as linear, [Abgrall et al. 2012, 2014; Dobrzynski and El Jannoun 2017; Xie et al. 2013], and (variants of) non-linear elasticity [Fortunato and Persson 2016; Moxey et al. 2016; Persson and Peraire 2009]. A comparison between different elasticity and distortion energies is presented in [Dobrev et al. 2019; Poya et al. 2016; Turner et al. 2016].

Direct and deformation methods have been tested on small collections of simple models, and, to the best of our knowledge, none of them can provide guarantees on the validity of the output or has been tested on large collections of models. There are also no reference implementations we could compare against.

Inversions and Intersections. Most of these methods introduce inverted elements during the curving of the high-order elements. Inverted elements can be identified by extending Jacobian metrics for linear elements [Knupp 2000, 2002] to high-order ones [Engvall and Evans 2018; Johnen et al. 2013; Peiró et al. 2014; Poya et al. 2016; Roca et al. 2012]. Various untangling strategies have been proposed, including geometric smoothing and connectivity modifications [Cardoze et al. 2004; Dey et al. 1999; Dobrev et al. 2019; Dobrzynski and El Jannoun 2017; Gargallo-Peiró et al. 2015; Gargallo Peiró et al. 2013; George and Borouchaki 2012; Geuzaine et al. 2015; Lu et al. 2013, 2014; Luo et al. 2008, 2002a; Peiró et al. 2008; Roca et al. 2012; Ruiz-Gironés et al. 2017, 2016a,b; Shephard et al. 2005; Stees and Shontz 2017; Steve L. Karman and Stefanski 2016; Toulorge et al. 2013, 2016; Turner 2017; Ziel et al. 2017]. None of these techniques can guarantee to remove the inverted elements.

An alternative approach is to start from an inversion-free mesh and slowly deform it [Persson and Peraire 2009; Ruiz-Gironés et al. 2017], explicitly avoiding inversions at the cost of possibly inaccurate boundary reproductions. These methods cannot, however, guarantee that the boundary will not self-intersect. Our approach follows a similar approach but uses a geometric shell to ensure element validity and prevention of boundary self-intersections.

[1] https://github.com/jiangzhongshi/bichon.git

Curved Optimal Delaunay Triangulation. [Feng et al. 2018] generalize optimal Delaunay triangulation paradigm to the high-order setting, through iteratively update vertices and connectivity. Their algorithm starts with a point cloud sampled from triangle meshes. However, the success of the method depends on the choice of final vertex number and sizing field, where insufficient vertices may result in broken topology or invalid tetrahedral meshes.

Software Implementation. Despite the large literature on curved mesh generation, there are very few implementations available.

Nektar [Moxey et al. 2018] is a finite element software with a meshing component, which can generate high-order elements. We do not explicitly compare as their documentation (Section 4.5.1.5 Mesh Correction[2]) states that the algorithm is not fully automatic and not designed to process robustly large collections of models. Gmsh [Geuzaine and Remacle 2009] is open-source software that supports the curved meshing of CAD models, but it does not support dense linear meshes as input. Despite the difference in the input type, we provide a comparison with Gmsh in Section 6.2, as it is the only method that we could run on a large collection of shapes.

To the best of our knowledge, the commercial software that support curved meshing (Pointwise [Pointwise 2018; Steve L. Karman and Stefanski 2016]) are also requiring a CAD model as input.

Animation. Curved tetrahedral meshes have also gained popularity in the context of fast animation. With fewer degrees of freedom and preserved geometric fidelity, [Mezger et al. 2007] observe the benefit of quadratic tetrahedra in the pipeline of physically-based animation. [Suwelack et al. 2013] further investigate the transfer problem when using curved meshes as a proxy.

2.2 Curved Structured Mesh Generation

The use of a hexahedral mesh as a discretization for a volume allows to naturally define C^k splines over the domain, which can be used as basis functions for finite element methods: this idea has been pioneered by Isogeometric Analysis (IGA), and it is an active research area. The generation of volumetric, high-order parametrizations that conform to a given input geometry is an extremely challenging problem [Peiró et al. 2015; Sorger et al. 2014]. Most of the existing methods rely on linear hexahedral mesh generation, which is on its own a really hard problem for which automatic and robust solutions to generate coarse meshes are still elusive [Gao et al. 2019; Guo et al. 2020; Li et al. 2012; Marschner et al. 2020; Palmer et al. 2020; Zhang et al. 2020] due to the inherently global nature of the problem. The current state of the art for IGA meshing is a combination of manual decomposition of the volume and the semi-automated geometrical fitting [Coreform 2020; Yu et al. 2020].

In contrast, our approach is automatic, i.e., we can automatically process thousands of models without any manual intervention while providing explicit guarantees on both the validity of the elements and the maximal geometric error. Its downside is the C^0 continuity of the basis on the elements' interfaces. However, we believe that curved tetrahedral meshing is a promising alternative as it dramatically simplifies both the meshing and fitting of high-order elements.

[2]https://doc.nektar.info/userguide/5.0.0/user-guidese17.html

2.3 Boundary Preserving Tetrahedral Meshing

We refer to [Hu et al. 2018] for a detailed overview of linear tetrahedral meshing, and we focus here only on the techniques that target boundary preserving tetrahedral meshing.

The most popular linear tetrahedral meshing methods are based on *Delaunay refinement* [Chew 1993; Ruppert 1995; Shewchuk 1998], i.e., the insertion of new vertices at the center of the circumscribed sphere of the worst tetrahedron in terms of radius-to-edge ratio. This approach is used in the most popular tetrahedral meshing implementations [Jamin et al. 2015; Si 2015], and, in our experiments, proved to be consistently successful as long as the boundary is allowed to be refined. A downside of these approaches is that a 3D Delaunay mesh, unlike the 2D case, might still contain "sliver" tetrahedra, thus requiring mesh improvement heuristics [Alliez et al. 2005; Cheng et al. 2000; Du and Wang 2003; Tournois et al. 2009]. [Alexa 2019] discusses this issue in detail and provides a different formulation to avoid it without the use of a postprocessing. [Alexa 2020] introduces an approach that does not allow insertion of Steiner points, making it not suitable for generic polyhedra domains.

There are many variants of Delaunay-based meshing algorithms, including *Conforming Delaunay tetrahedralization* [Cohen-Steiner et al. 2002; Murphy et al. 2001], *constrained Delaunay tetrahedralization* [Chew 1989; Shewchuk 2002; Si and Gärtner 2005; Si and Shewchuk 2014], and *Restricted Delaunay tetrahedralization* [Boissonnat and Oudot 2005; Cheng et al. 2008; Engwirda 2016].

To the best of our knowledge, all these methods are designed to allow some modifications of the input surface (either refinement, resampling, or approximation). One exception is the constrained Delaunay implementation in TetGen [Si 2015] that allows disabling any modification to the boundary. However, this comes at the cost of much lower quality and potential robustness issues, as we show in Appendix C.

A different tetrahedral meshing approach has been proposed in [Hu et al. 2018], and its variants [Hu et al. 2019, 2020], where the problem is relaxed to generate a mesh that is close to the input to increase robustness. However, these approaches are not directly usable in our setting, as we require boundary preservation.

Due to these issues, we propose a novel boundary-preserving tetrahedral meshing algorithm specifically tailored for the shell mesh generated by our curved meshing algorithm.

2.4 Curved Surface Fitting

There are many algorithms for fitting curved *surfaces* to dense 3D triangle meshes. The most popular approaches fit spline patches, usually on top of a quadrangular grid. Since generating quadrilateral meshes is a challenging problem for which robust solutions do not exist yet, we refer to [Bommes et al. 2012] for an overview, and only review in this section algorithms for unstructured curved mesh generation, which are more similar to our algorithm. We note that the focus of our paper is volumetric meshing: while we generate an intermediate curved surface mesh, this is not the goal of our algorithm, especially since the generated surface is only C^0 on edges.

[Hoppe et al. 1994] fits a smooth surface represented by a point cloud to a curved triangle mesh based on a subdivision surface scheme and an interleaving mesh simplification and fitting pipeline

that preserves sharp features. The algorithm does not provide explicit correspondence to the input: they are defined using distance closest point, which is not bijective far from the surface.

[Krishnamurthy and Levoy 1996] converts dense irregular polygon meshes of arbitrary topology into coarse tensor product B-spline surface patches with accompanying displacement maps. Based on the work [Lin et al. 2007] that fits triangle surface meshes with Bézier patches, [Zhang et al. 2011] fits triangle surface meshes with high-order B-spline quadrilateral patches and adaptively subdivide the patches to reduce the fitting error. These methods produce smooth surfaces but do not have feature preservation.

Another related topic is the definition of smooth parametric surfaces interpolating triangle meshes. We refer to [Zorin 2000] for an overview of subdivision methods and discuss here the approaches closer to our contribution.

[Hahmann and Bonneau 2003] proposed to use triangular Bézier patches to define smooth surfaces over arbitrary triangle meshes ensuring tangent plane continuity by relaxing the constraint of the first derivatives at the input vertices. Following Hahmann's work, [Yvart et al. 2005b] presents a complete pipeline: perform QEM simplifications, trace the coarse mesh onto the dense one and perform parameterization relaxing and smoothing. Then it fits a hierarchical triangular spline [Yvart et al. 2005a] to the surface. More recent work [Tong and Kim 2009] approximates the triangulation of an implicit surface with a G^1 surface. These schemes are usually designed to interpolate existing meshes rather than simplifying a dense linear mesh into a coarse curved mesh and are thus orthogonal to our contribution.

3 SHELL PRELIMINARIES

We briefly overview [Jiang et al. 2020] as our work uses and extends it. [Jiang et al. 2020] introduces bijective projection shells (which we will abbreviate as shells in the rest of the paper), a new geometry processing tool to perform mesh editing while preserving a close-by bijective map to the input surface.

Shell. The shell is defined by three trianglulated surface meshes $\overline{\mathcal{S}} = \{(B_s, V_s, T_s), F_s\}$ sharing the same mesh connectivity F_s, where B_s, V_s, and T_s are the vertices of the bottom, middle, and top surface respectively. Each triangle in F_s corresponds to a *generalized prism*, defined by connecting the corresponding triangles in the three surfaces with straight edges, called *pillars*. Each prism P has three vertices $v_i \in V_s, t_i \in T_s, b_i \in B_s, i = 1, 2, 3$ on the middle, top, and bottom surface, respectively. Each pillar decomposes into a top $h_i^T = t_i - v_i, i = 1, 2, 3$ and bottom $h_i^B = b_i - v_i, i = 1, 2, 3$ slab, and each slab can be canonically decomposed into 3 tetrahedra, and each tetrahedron contains a constant vector field aligned with the pillars it is connected with [Jiang et al. 2020, Figure 4]. The vector field is used to define a *projection operator* Π within the shell.

Projection Operator. For every prism P, the projection operator Π_P is defined as the tracing of the piecewise constant vector field V inside the decomposed prism, by assigning to each tetrahedron $T_j^P, j = 1, \ldots, 6$, the constant vector field defined by the only edge of T_j^P which is one of the oriented pillars $h_i^T, h_i^B, i = 1, 2, 3$ ([Jiang et al. 2020, Figure 4]). That is, $\forall p \in T_j^P$

$$\Pi_P(p) = h_i^k,$$

where i is the index of the vertex corresponding to the pillar edge of T_j^P, and k is either the top or bottom surface. The shell projection operator Π is defined as the operator whose restriction to P, $\Pi|_P$ is Π_P, and it defines a bijective map between every pair of specific triangle meshes contained in the shell, called *sections*.

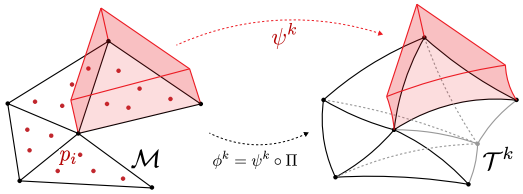


Fig. 2. Input triangle mesh $\mathcal{M}$ and points $\mathcal{P}$. Output curved tetrahedral mesh $\mathcal{T}^k$ and bijective map ϕ^k.

Section. A triangle mesh is a section if it is contained within the shell, and if the dot product of the normal of each of its triangles with the vector field in each of the overlapping tetrahedra is positive. The projection operator Π defines a bijective map between any pair of sections if the shell is *valid*.

Shell Validity. [Jiang et al. 2020] defines a shell $\overline{\mathcal{S}}$ to be *valid* with respect to an input mesh $\mathcal{M}$ if it satisfies two conditions:

(1) The volumes of all possible tetrahedral decomposition of a prism (24 of them) are positive.
(2) $\mathcal{M}$ is a section for all possible tetrahedral decompositions. That is, the input mesh is contained within the shell, and the dot product between the mesh's normals and the shell's pillar is positive.

Singularity. Singular vertices are a special geometric configuration, where the neighboring triangles of a specific vertex admit a conflicting set of normals. [Jiang et al. 2020] extends the shell construction to allow such cases. Around the (isolated) singular vertices, the prisms are pinched to become generalized pyramids, composed of two tetrahedra instead of three.

The algorithm to build the shell creates an initial valid extrusion, potentially thin and dense, and then iteratively uses the shell local operations (i.e., vertex smoothing, edge collapse, edge split, and edge flip) [Jiang et al. 2020, Section 3.4] to improve its quality while preserving the validity.

3.1 Variation from the Original Algorithm

To extend the shell formulation in [Jiang et al. 2020] to accommodate for feature preservation (Section 5), we modify the definition of a valid section [Jiang et al. 2020, Definition 3.1] by relaxing several zero-measure intersections between a triangle and a prism in the discrete case. That is, we do not consider the prism to be intersecting a triangle if they share only one vertex of the triangle; we also ignore when the prism and the triangle intersect only on one feature edge if they are on the opposite sides of the edge. The bijectivity and validity condition of the shell projection trivially holds.

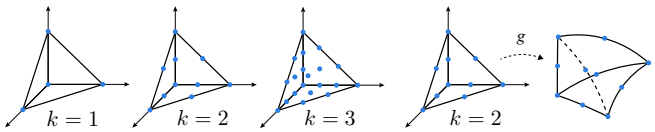


Fig. 3. Lagrange nodes on the reference element $\hat{\tau}$ for different $k = 1, 2, 3$ and example of geometric mapping g.

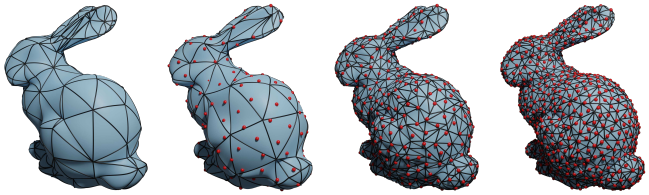


Fig. 4. Effect of the choice of the set $\mathcal{P}$ on the output.

4 CURVED TETRAHEDRAL MESH GENERATION

Input. The input of our algorithm is a collection of oriented manifold, watertight, self-intersection-free triangle mesh $\mathcal{M} = (V, F)$, and a set of points $p_i \in \mathcal{P}$ (possibly empty) on the surface of $\mathcal{M}$ (Figure 2, left) where the distance bound ε is prescribed. The collection $\mathcal{M}$ must be consistently oriented such that it is the boundary of an oriented 3-manifold. A set of edges can also be optionally provided as annotated features (Section 5).

Output. The output of our algorithm is a tetrahedral mesh $\mathcal{T}^k = (V^k, T^k)$ of order k. Formally, each tetrahedron $\tau \in T^k$ is defined through the *geometric map* from the reference tetrahedron $\hat{\tau}$,

$$g^\tau = \sum_{j=1}^{n} c_j^\tau l_j(\hat{u}, \hat{v}, \hat{w}), \tag{1}$$

where $\hat{u}, \hat{v}, \hat{w}$ are the local coordinates of a point in $\hat{\tau}$, c_j^τ are the control points for a tetrahedron τ, and l_j are polynomial bases (typically Lagrange bases). For two tetrahedra τ_1 and τ_2 of T^k sharing a face F, the restriction of the maps $g^{\tau_i}|_F$, $i = 1, 2$ coincide. Figure 3 shows the position of the control points c_j on the reference element for $k = 1, 2, 3$ for the Lagrange bases. We call the tetrahedralization of a curved mesh $\mathcal{T}^k$ *positive* if the Jacobian determinant of g^τ $\det(J_{g^\tau})$ is positive everywhere on every τ. In particular, for $k = 1$, since g is affine, J_{g^τ} is constant and the positivity reduces to the positive orientation of the vertices [Shewchuk 1997].

Note that, while bijectivity of the geometric map g^τ implies positivity, the reverse is not true. Therefore, our algorithm not only checks for $\det(J_{g^\tau}) > 0$, but also ensures that the boundary $\partial\mathcal{T}^k$ does not intersect; we show in Appendix A that these two conditions guarantee the bijectivity of g^τ. Furthermore, our algorithm ensures that the distance from any point in $\mathcal{P}$ to $\partial\mathcal{T}^k$ (the surface of $\mathcal{T}^k$) is smaller than a user-controlled parameter ε.

We are not assuming anything on $\mathcal{P}$: a sparse set of points will generate a mesh that is less faithful to the input geometry, while a dense sampling computed, for instance with Poisson disk sampling [Bowers et al. 2010], will prevent the surface from deviating too much (Figure 4).

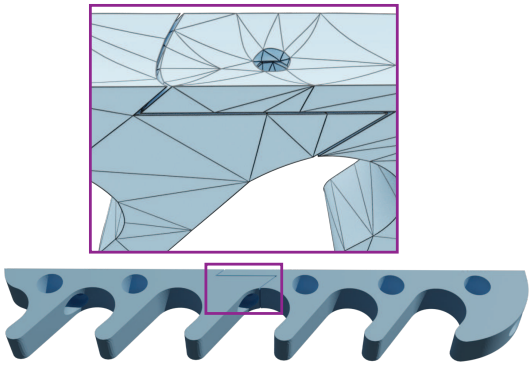


Fig. 5. Our algorithm maintains free of intersection even on challenging models, without the need of setting an adaptive threshold.

Our algorithm guarantees that the tetrahedralization is positive and that $\partial\mathcal{T}^k$ does not have self-intersections. It also aims at coarsening $\mathcal{T}^k$ as much as possible while striving to obtain a good geometric quality. To reliable fit $\partial\mathcal{T}^k$ to $\mathcal{M}$, we require a bijective map

$$\phi^k : \mathcal{M} \to \partial\mathcal{T}^k$$

from the input $\mathcal{M}$ to the surface of $\mathcal{T}^k$ (Figure 2 right). Our algorithm also generates this map and exposes it as an output for additional uses, such as attribute transfer. Note that, since we build upon the shell construction in [Jiang et al. 2020], we also guarantee $\partial\mathcal{T}^k$ is homeomorphic and topology-preserving with respect to $\mathcal{M}$ (Figure 5).

> To simplify the explanation, we use the bar $\overline{\ }$ to represent quantities on the *straight* linear shell, the tilde $\widetilde{\ }$ for the *curved* shell, and hat $\hat{\ }$ for the reference elements (e.g., $\overline{P}$ is the prism on the straight coarse shell, $\widetilde{P}$ is the curved prism, and $\hat{P}$ is the prism on the reference configuration).

Definition 4.1. We call a curved mesh $\mathcal{T}^k$ and its boundary mapping ϕ^k to $\mathcal{M}$ *valid* if it satisfies the following conditions:

(1) ϕ^k is bijective;
(2) the distance between any $p \in \mathcal{P}$ and $\partial\mathcal{T}^k$ is less than ε;
(3) $\mathcal{T}^k$ is positive (i.e., each geometric map g^τ has a positive Jacobian determinant).

Overview. Our algorithm starts by creating a valid mesh (i.e., it satisfies 4.1), then it performs local operations (Appendix B) to improve $\mathcal{T}^k$ (i.e., coarsen it and improve its quality) while ensuring all the conditions remain valid with respect to local modification. To achieve this goal, our algorithm uses two stages: (1) curved shell generation and (2) tetrahedral mesh generation and optimization.

Stage 1: Curved Shell Construction. In the first stage (Section 4.1) we extend the shell construction of [Jiang et al. 2020] by combining

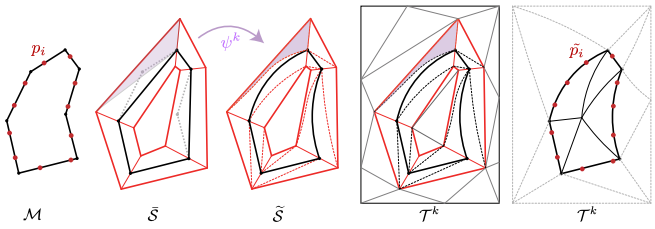


Fig. 6. Overview of curved mesh generation pipeline.

the shell projection Π with a high-order volumetric mapping

$$\psi^k : \overline{\mathcal{S}} \to \widetilde{\mathcal{S}}.$$

We start form the *valid* shell $\overline{\mathcal{S}}$ constructed from the input mesh $\mathcal{M}$ (i.e., $\mathcal{M}$ is a section $\overline{\mathcal{S}}$). We call $\overline{\mathcal{S}}$ a *projection shell* and call the prismatic projection Π. Together with the construction of $\overline{\mathcal{S}}$, we build an order k *curved prismatic shell* $\widetilde{\mathcal{S}}$ that defines a curved layer around $\mathcal{M}$ and a *bijective* map ψ^k between $\widetilde{\mathcal{S}}$ and $\overline{\mathcal{S}}$ (Figure 6, first three figures) that ensures that the distance between $\mathcal{M}$ and $\partial\mathcal{T}^k$ is smaller than ε (Section 4.2). That is, $\phi^k(p) < \varepsilon$ for any $p \in \mathcal{P}$. (Note that we do not require $\mathcal{M}$ to be a section of $\widetilde{\mathcal{S}}$.)

To facilitate the volumetric meshing in the next stage (Section 4.3), we restrict the top and bottom surface of $\widetilde{\mathcal{S}}$ to be linear (independently from the order of ψ^k). The final output of this first stage is a high-order volumetric shell, a bijective mapping $\phi^k = \psi^k \circ \Pi$, and a *positive* tetrahedralization of $\widetilde{\mathcal{S}}$ with flat boundary. In other words, the tetrahedralization of $\widetilde{\mathcal{S}}$ satisfies 4.1.

Stage 2: Tetrahedral Mesh Generation. In the second stage (Section 4.3) we use boundary-conforming tetrahedralization to connect the top and bottom surface of $\widetilde{\mathcal{S}}$ with a background tetrahedral mesh, thus generating a *positive* order k tetrahedralization $\mathcal{T}^k$ of a bounding box around the input, which we can further optimize with local operations to improve its quality (Figure 6, last two figures).

To ensure that our first condition is satisfied, we define the mapping ϕ^k as a composition of several mappings, which we ensure are bijective. For the second condition, we initialize our construction with ϕ^k as the identity, and thus, the distance at the sample points is zero. After every operation, we recompute the distance and "undo" the operation if the distance becomes larger than ε. To ensure that the last condition holds, we rely on checking if all prisms (linear and curved) have positive geometric mapping, which ensures that they can be tetrahedralized with a positive tetrahedralization. Ensuring the condition while coarsening $\mathcal{M}$ allows us to generate a coarse *curved* tetrahedral mesh $\mathcal{T}^k$ and the *bijective* map ϕ^k to the input mesh $\mathcal{M}$.

4.1 High-order Shells

To simplify the explanation, we first focus on the case where $\varepsilon = \infty$, that is, we aim at generating an as-coarse-as-possible curved mesh. Note that, the trivial solution (i.e., a single tetrahedron) is not necessary a valid $\mathcal{T}^k$ since it would be impossible to build the bijective mapping ϕ^k.

The output of [Jiang et al. 2020] is a coarse shell $\overline{\mathcal{S}}$ with a piecewise linear middle surface. To curve it, we construct a shell $\widetilde{\mathcal{S}}$ and the bijective map ψ^k while constructing $\overline{\mathcal{S}}$. The shell $\widetilde{\mathcal{S}}$ is constructed warping every prism $\widetilde{P}$ of $\widetilde{\mathcal{S}}$ with ψ^k. Since we define ϕ^k as $\psi^k \circ \Pi$, and Π satisfies the first two conditions 4.1, we only need to ensure that ψ^k is bijective, for ϕ^k to be bijective. We define the mapping $\psi^k = \widetilde{\omega}^k \circ (\overline{\omega})^{-1}$ through two parametrization maps from the reference prism $\hat{P}$:

$$\overline{\omega} : \hat{P} \to \overline{P}, \qquad \widetilde{\omega}^k : \hat{P} \to \widetilde{P}.$$

Both mappings $\overline{\omega}$ and $\widetilde{\omega}^k$ are defined as the tensor product between the base triangular mapping (high-order for $\widetilde{\omega}^k$) and pillar's barycentric heights. For a prism $\widetilde{P}$, piecewise defined for top slab and bottom slab,

$$\begin{aligned}\widetilde{\omega}^k(\hat{u},\hat{v},\hat{h}) = &\sum_{j=1}^{n} c_j l_j^k(\hat{u},\hat{v})(|2\hat{h}-1|+1)\\ &+\sum_{j=1}^{3} b_j l_j^1(\hat{u},\hat{v})\max(1-2\hat{h},0)\\ &+\sum_{j=1}^{3} t_j l_j^1(\hat{u},\hat{v})\max(2\hat{h}-1,0)\end{aligned}$$

where $\hat{u},\hat{v},\hat{h}$ are the barycentric coordinates in the reference prism, c_j the control points of the middle triangle, t_j and b_j the top/bottom triangles' vertices, and l_j^k is an order k triangle polynomial basis. Note that for the top/bottom part we use only t_j and b_j as they remain linear. By ensuring that ψ^k is bijective, we guarantee that any curved tetrahedralization of a prism $\widetilde{P}$ will be a valid tetrahedralization of $\widetilde{\mathcal{S}}$.

We note that to decouple the following tetrahedral mesh generation and the curved shell generation, we ensure that $\widetilde{\omega}^k$ maps the top and bottom face of the curved prism $\widetilde{P}$ to a linear triangle.

After each local operation, we generate samples $\hat{s}_i, i = 1,\dots,m$ on the parametric base of the prism $\hat{P}$ and use $\Pi^{-1} \circ \overline{\omega}$ to map $\hat{s}_i$ back to $\mathcal{M}$ and $\widetilde{\omega}^k$ to map them to $\partial\mathcal{T}^k$. Using the mapped points we solve

$$\min_{c_i} \sum_{i=1}^{m} \|(\Pi^{-1} \circ \overline{\omega})(\hat{s}_i) - \widetilde{\omega}[c_i]^k(\hat{s}_i)\|_2^2,$$

where c_i are the control points of $\widetilde{\omega}^k$. As $\widetilde{\omega}[c_i]^k(\hat{s}_i)$ is a linear function of c_i. This is a quadratic optimization problem. The control points of the top and bottom surface are fixed to ensure that $\widetilde{\omega}^k$ maintains the two surfaces as linear. We validate the bijectivity of $\widetilde{\omega}^k$ by checking positivity of the determinant [Johnen et al. 2013] after splitting into tetrahedral elements [Moxey et al. 2015] and that the top and bottom surfaces are intersection free. The intersection is simplified in our case since fast and exact algorithms[Guigue and Devillers 2003] are available since the top and bottom surfaces stay linear.

4.2 Distance Bound

In the previous section, we explained how to generate a curved shell $\widetilde{\mathcal{S}}$ that satisfies 4.1. To ensure that the middle surface of $\widetilde{\mathcal{S}}$ has a controlled distance from the points in $\mathcal{P}$, we interleave a distance

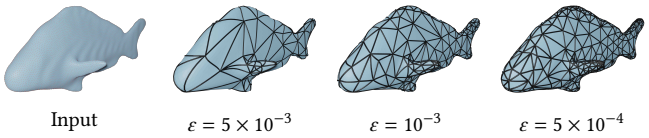


Fig. 7. A model simplified with different distances.

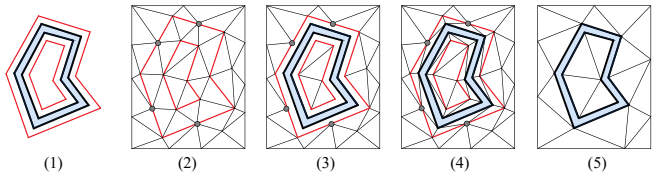


Fig. 8. Two dimensional overview of the five steps of our boundary preserving tetrahedral meshing algorithm.

check in the construction of $\widetilde{\omega}^k$ after each local operation. Formally, after every local operation we use the mapping ϕ^k to map every point $p_i \in \mathcal{P}$ to $\widetilde{p}_i = \phi^k(p_i)$ a point on the coarse curved middle surface of $\widetilde{\mathcal{S}}$ and, if $\|p_i - \widetilde{p}_i\| \geq \varepsilon$ we reject the operation. The initial shell is trivially a valid initialization as ϕ^k is identity and thus, the distance is zero. Note that, $\widetilde{p}_i$ is not necessarily the closest point to p_i on $\partial\mathcal{T}^k$, thus $\|p_i - \widetilde{p}_i\|$ is an upper bound on the actual pointwise distance. Figure 7 shows the effect of the distance bound on the surface; a small distance will lead to a denser mesh with more details, while a large one will allow for more coarsening.

4.3 Tetrahedral Meshing

The outcome of the previous stage is a curved tetrahedralization of $\widetilde{\mathcal{S}}$ that closely approximates $\mathcal{M}$ with linear ("flat") boundaries. We now consider the problem of filling its interior (and optionally its exterior) with a tetrahedral mesh, a problem known as *conforming boundary preserving* tetrahedralization.

Several solution exists for this problem (Section 2.3) and the most common implementation is TetGen [Si 2015]. Most algorithms refine the boundary, which allows deriving bounds on the quality of the tetrahedral mesh. However, in our setting, this is problematic, as any change will have to be propagated to the curved shell. To avoid coupling the volumetric meshing problem with the curved shell coarsening, while technically possible it is very challenging to implement robustly, we opt for using a tetrahedral meshing algorithm that preserves the boundary exactly. Not many algorithms support this additional constraint, the only one with a public implementation is the widely used TetGen algorithm. However, we discovered that, when this option is used, it suffers from robustness issues, which we detail in Appendix C. To solve this problem in our specific setting, we propose in the following five step algorithm (Figure 8) taking advantage of the availability of a shell, based on the TetWild [Hu et al. 2018] algorithm.

Step 1. To generate a boundary preserving linear mesh, we first exploit the shell to extrude the bottom surface B (and top T) further by a positive (potentially small) constant δ such that the newly extruded bottom surface B_e (and top T_e) does not self-intersect. The space between B and B_e (and between T and T_e) consists of prisms divided into positive tetrahedra.

Step 2. Then we insert B_e and T_e in a background mesh $\mathcal{B}$ generated following TetWild algorithm ([Hu et al. 2018, Section 3.1]), that is, we use the triangle of B_e and T_e as the input triangle meshes for the first stage of the TetWild algorithm, which inserts them into a background mesh $\mathcal{B}$, so that each input triangle is a union of faces of refinement of $\mathcal{B}$.

We interrupt the algorithm after the binary space partitioning (BSP) subdivision (and before the TetWild mesh optimization [Hu et al. 2018, Section 3.2]) to obtain a positive tetrahedral mesh in rational coordinates with a surface with the same geometry of B_e and T_e, but possibly different connectivity as TetWild might refine it during the BSP stage.

Step 3. Our original goal was to compute a mesh conforming to B and T, but we could not do it directly with TetWild as they might be refined. We now replace the mesh generated by TetWild between B_e and T_e with another one conforming to B and T. To achieve this, we delete all tetrahedra between B_e and T_e, and insert the surfaces B and T, which will "float" in the empty space between B_e and T_e. We now want to fill the space between B and B_e with positive tetrahedra conforming to the surfaces B and T.

Step 4. Every prism P, made by a bottom triangle B^T and a bottom extruded triangle B_e^T and its corresponding bottom extruded refined triangle $B_e'^T \in \mathcal{B}$, can be tetrahedralized without refining B: That is, we first decompose the prism B^T, B_e^T in tetrahedra (always possible by construction), then refine every tetrahedron touching $B_e'^T$. By repeating the same operation on the space between T and T_e we will have a positive linear boundary conforming tetrahedral mesh of B and T.

Step 5. The tetrahedra generated in the previous step will have rational coordinates and will also likely have low quality. To round the coordinates to floating-point representation and to improve their quality, we use the mesh optimization stage of TetWild, with the minor variant of keeping the vertices and edges on B and T frozen. Note that the vertices in B and T are already roundable to floating-point representation, as they were part of the input.

Curved Tetrahedral Mesh Optimization. After generating the conforming linear tetrahedral mesh. we stitch it with the tetrahedralized $\widetilde{\mathcal{S}}$ to obtain a valid output mesh $\mathcal{T}^k$ (Definition 4.1). However, its quality might be low, in particular in the curved region, as $\widetilde{\mathcal{S}}$ can be thin with large triangles. To improve the quality of $\mathcal{T}^k$ we adapt the local operation of a linear pipeline to our curved settings. We propose three local operations: smoothing, collapse, and flip. Since the surface of $\mathcal{T}^k$ is already coarse and of high quality, as part of the definition of ϕ^k, we prevent any local operation from changing it. We validate every local operation (i.e., check the positivity of $\mathcal{T}^k$) using the convex-hull property [Johnen et al. 2013] and reject the operation if it is violated. Our local operations are prototypical, and we leave as future work a more comprehensive study of curved mesh optimization.

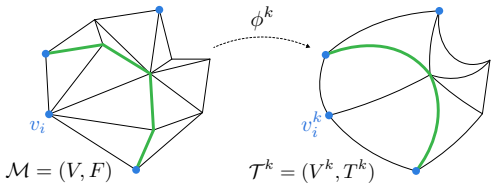


Fig. 9. Input triangle mesh with features and output curved mesh with feature preserved equipped with bijective map ϕ^k.

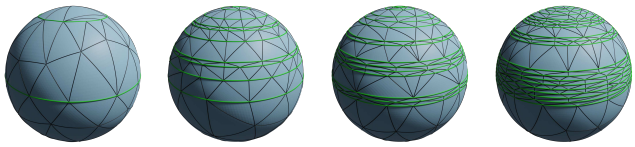


Fig. 10. A sphere with different marked features (green). As we increase the number of features, our algorithm will preserve them all but the quality of the surface suffers.

Smoothing. As for the linear case, we compute the total energy of a vertex v by summing up the energies of the tetrahedra adjacent to it, which we compute on 56 uniformly sampled points. We then perform gradient descent for all high-order nodes in the star of v [Arnold et al. 2000; Farrell et al. 2019; Vanka 1986]. That is, we collect all edge nodes, face nodes, and cell nodes of the one-ring neighborhood of v. Differently from the linear case, the optimization is expensive since the nodes neighborhood typically contains hundreds of nodes.

Collapse and Swap. The collapse and swap are the same as in a linear mesh, and we place the high-order nodes of the newly created face on the linear flat face.

5 FEATURE PRESERVING CURVED SHELL

Input. We enhance the input to additionally include a set of feature edges $f_i \in \mathcal{F}$ and feature vertices $v_i \in \mathcal{V}$ such that no triangle in F has more than one feature edge (Figure 9 left). (This property can be satisfied on any generic mesh by performing 1-to-3 refinement on every triangle with more than one feature edge).

Output. Since the input has features, the output curved mesh $\mathcal{T}^k$ will also have curved feature edges $f_i^k \in \mathcal{T}^k$, feature vertices $v_i^k \in \mathcal{V}^k$, and the bijective map ϕ^k preserves features by bijectively mapping $\mathcal{F}$ to $\mathcal{F}^k$ and $\mathcal{V}$ to $\mathcal{V}^k$. Our method makes no assumption on the topology and "quality" of the features. If the features are reasonable, it will produce a high-quality mesh, while if the features are close, our algorithm will preserve them and result in smaller triangles on the surface. (Figure 10).

The previous construction generates valid curved tetrahedral meshes and the bijective map ϕ^k based on the construction of [Jiang et al. 2020]. However, the shell construction cannot coarsen features: the authors suggest freezing them. For instance, when performing an edge collapse on the feature, the new coarse edge (orange) will not map to the feature (green) anymore (Figure 11). To ensure feature preservation we extend Definition 4.1.

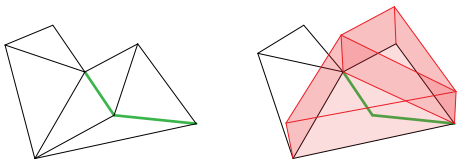


Fig. 11. Input feature (green) is not preserved after traditional shell simplification.

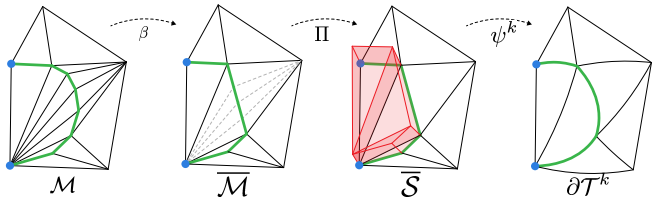


Fig. 12. Overview of the construction of the first stage of our pipeline.

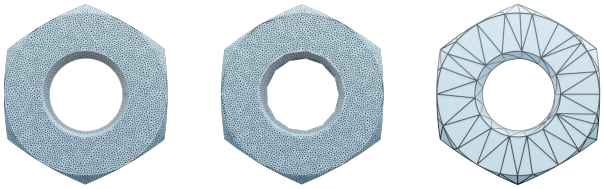


Fig. 13. The input mesh has feature edges snapped, to create a valid shell, as well as the curved mesh.

Definition 5.1. We call a curved mesh $\mathcal{T}^k$ and its mapping ϕ^k from $\mathcal{M}$ *valid and feature preserving* if they are valid (Definition 4.1) and ϕ^k bijectively maps $\mathcal{F}$ to $\mathcal{F}^k$ and $\mathcal{V}$ to $\mathcal{V}^k$

As for the non-feature preserving case, we always aim to maintain a valid feature preserving $\mathcal{T}^k$.

To account for features, we propose to change the prismatic map Π, that is, we only need to change the first stage. This is done by snapping the input features (Section 5.1). That is, we modify $\mathcal{M}$ to "straighten" the feature to ensure that the coarse prismatic projection preserves them and construct a mapping β between the straight mesh $\overline{\mathcal{M}}$ and $\mathcal{M}$ (Figure 13).

The outcome is a *valid* shell $\overline{\mathcal{S}}$ with respect to the straight surface $\overline{\mathcal{M}}$ (i.e., $\overline{\mathcal{M}}$ is a section $\overline{\mathcal{S}}$) that preserve features, the prismatic projection Π, and the bijective map β that can be directly used in the curved pipeline (Section 4). That is, the mapping ϕ^k will be defined as $\phi^k = \psi^k \circ \Pi \circ \beta$.

As for the non-preserving feature pipeline, we ensure that our conditions are always met, starting from a trivial input and rejecting operations violating them. Our goal is to modify the input mesh $\mathcal{M}$ and create $\overline{\mathcal{M}}$ by moving its vertices. In such a way, the mapping β is simply barycentric. To guarantee bijectivity of ϕ^k we need to ensure that all mappings composing it are bijective, in particular β. To ensure that β is bijective, it is enough that $\overline{\mathcal{M}}$ is self-intersection

Fig. 14. The input edges feature (green) are grouped together in poly-lines and categorized in graph (left) and loops (right). For every graph, we add the nodes (blue) to the set of feature vertices.

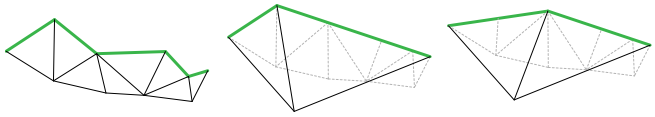


Fig. 15. Illustration of smoothing on a feature.

free (guaranteed by the shell construction) and that all its triangles have positive areas. By straightening the features of $\mathcal{M}$ we ensure that the edges of the prism will map to the feature. Thus, ϕ^k will be feature preserving.

Feature Grouping. The first step of our pipeline consists of grouping successive edges $f_i \in \mathcal{F}$ into poly-lines and identifying two categories: loops and graphs (Figure 14). For every graph, we identify its nodes and add them as feature vertices. In other words, we add to $\mathcal{V}$ all the end-points and junction of poly-lines.

5.1 Feature straightening.

To allow feature coarsening, we propose to straighten $\mathcal{M}$ to ensure that all features are collinear. In other words, we build, together with the shell $\overline{\mathcal{S}}$, a mesh $\overline{\mathcal{M}} = (\overline{V}, F)$ (i.e., a mesh with the same connectivity F of $\mathcal{M}$) and features $\overline{\mathcal{F}}$ such that every triangle of $\overline{\mathcal{M}}$ has a positive area, $\overline{\mathcal{M}}$ is a section of $\overline{\mathcal{S}}$, and the features in $\overline{\mathcal{F}}$ are collinear. In such a way, the mapping β is trivially defined as piecewise affine ($\overline{\mathcal{M}}$ and $\mathcal{M}$ share the same connectivity) and is locally injective as long as all the triangles on $\overline{\mathcal{M}}$ have positive areas. Note that the bijectivity of β follows from the fact that the shell prevents self-intersections of $\overline{\mathcal{M}}$.

To construct a mesh $\overline{\mathcal{M}}$ with straight features, we start with $\overline{\mathcal{M}} = \mathcal{M}$ (in the beginning, all prisms of $\overline{\mathcal{S}}$ cover at most one feature edge). Let $\overline{f}^1 = \{\overline{f}_i^1\}, i = 1, \dots, n$ and $\overline{f}^2 = \{\overline{f}_i^2\}, i = 1, \dots, k$ two chains of feature edge belonging to the same feature $\overline{f} \in \overline{\mathcal{F}}$. For every local operation acting on a feature $\overline{f}_1$ and $\overline{f}_2$ we first construct the new feature $\overline{f}^n = \{\overline{f}_i^n\}, i = 1, \dots, k$ such that the segments $(\overline{f}_i^n, \overline{f}_{i+1}^n)$ are collinear and their length is proportional to (f_i, f_{i+1}) (the feature vertices in the input mesh $\mathcal{M}$), that this we use arc-length cross parameterization from f to $\overline{f}^n$ (Figure 15 show an example of smoothing a feature). Moving vertices of $\overline{f}^n$ will also move the vertices of $\overline{\mathcal{M}}$ thus, straighten the mesh as the local operations proceed. After the construction of $\overline{f}^n$ we check if the newly constructed $\overline{\mathcal{M}}$ is still a section of $\overline{\mathcal{S}}$ and if the triangles modified by the straightening have areas larger than ϵ. In practice, we choose

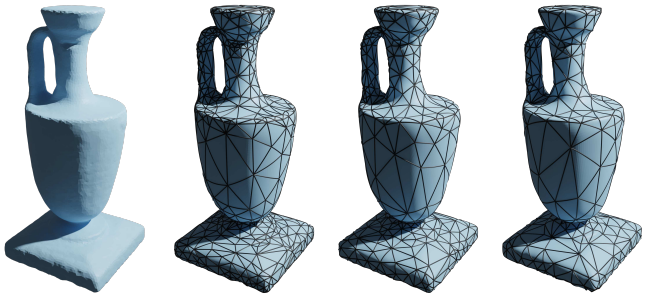


Fig. 16. Curved meshes of different order. The additional degrees of freedom allow for more coarsening.

$\epsilon = 10^{-10}$: a smaller value would lead to numerical instabilities and a larger one to less straightening.

Note that not all features can be straightened: for instance, if a triangle has three feature vertices (the snapped feature will result in a degenerate triangle, thus, β will not be bijective) or if the snapping flips the normal ($\overline{\mathcal{M}}$ will no longer be a section of $\overline{\mathcal{S}}$). Both are extremely rare cases in our dataset.

6 RESULTS

Our algorithm is implemented in C++, using Eigen [Guennebaud et al. 2010] for the linear algebra routines, CGAL [The CGAL Project 2020] and Geogram [Lévy 2015] for predicates and geometric kernel, libigl [Jacobson et al. 2016] for basic geometry processing routines, and meshio [Schlömer 2020] for converting across the different formats. We run our experiments on cluster nodes with Intel Xeon Platinum 8268 CPU 2.90GHz. The reference implementation and the data used to generate the results will be released as an open-source project.

To simplify the exposition, all meshes presented in this section are quartic meshes ($k = 4$). Our method is flexible and, for lower k, it will generate denser meshes (Figure 16).

6.1 Large Scale Validation.

We tested the robustness and quality of the result produced by our algorithm on three datasets: (1) Thingi10k dataset [Zhou and Jacobson 2016] containing 3574 models without features; (2) the first chunk of the ABC dataset [Koch et al. 2019] with 5328 models with features marked from the STEP file and (3) the CAD dataset [Gao et al. 2019] containing 106 models with semi-manual features.

Note that the original datasets contain more models since, for each of them, we selected meshes satisfying our assumptions: intersection-free (using the same strategy as in [Jiang et al. 2020] with a distance tolerance of 10^{-6} and dihedral angle of $2°$) oriented, manifold triangle meshes, smallest triangle area larger than 10^{-8}.

Our method has only the geometry accuracy parameter ε, which we set to 1% of the longest bounding box edge, and the point set $\mathcal{P}$ which we set as the input vertices V. With this basic setup, our algorithm aims to produce the coarsest possible mesh while preserving features and striving to generate high-quality meshes. Our algorithm successfully generates curved meshes for 3527 for Thingi10k,

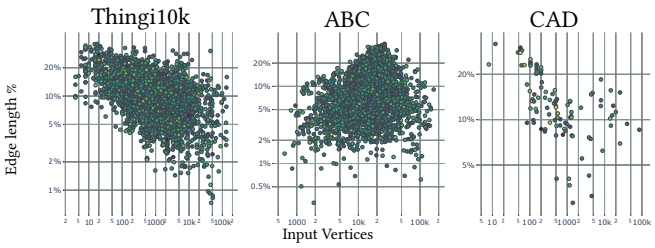


Fig. 17. Relative average edge length (with respect to longest bounding box edge of each model) of our curved meshes versus number of input vertices.

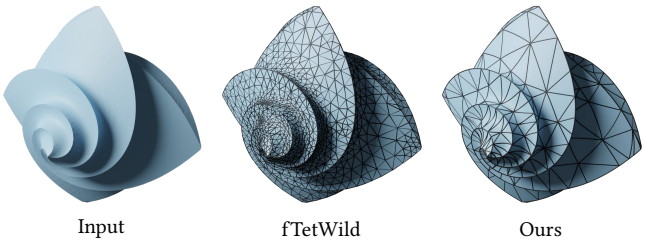


Fig. 18. Within the same distance bound (10^{-3} of the longest bounding box side), our method generates a coarser high order mesh, compared to the linear counterpart generated by fTetWild.

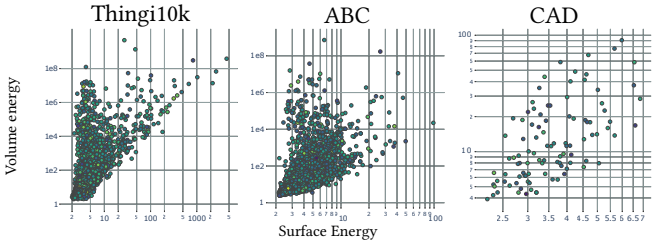


Fig. 19. Surface and volume average MIPS energy of the output of our method (the CAD volume energy is truncated at 100, excluding 6 models).

5268 for the ABC, and all for CAD dataset within 12 hours; by allowing more time, all models but 3 can be successfully processed. The 3 failures are due to models with a small one-tetrahedra component that "move" inside the shell as it grows. This is an implementation choice: we use collision detection instead of continuous collision detection for efficiency reasons.

Our method successfully generates coarse meshes whose average edge length is 10% of the model size while preserving features (Figure 17). Figure 18 shows how our method successfully captures the features and coarsen the surface with curved elements, while many linear elements (generated with fTetWild [Hu et al. 2020]) are required to closely approximate the surface.

The output of our algorithm can be directly used in the simulation (Section 6.4) since we guarantee that the geometric mapping g is positive. To ensure good conditioning and performance of the numerical solver, we measure the MIPS energy [Fu et al. 2015; Hormann and Greiner 2000] of our output meshes (Figure 19).

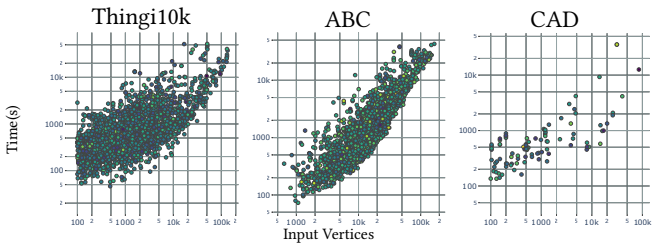


Fig. 20. Timing of our algorithm versus the input number of vertices.

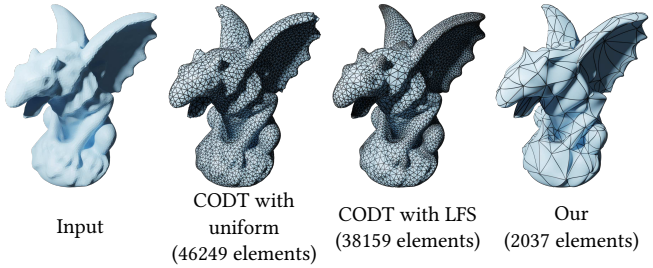


Fig. 21. Compared with Curved ODT, our method does not rely on setting vertex number and sizing field and can generate coarse valid results.

Figure 20 shows the running time of our method with respect to the number of vertices. The running time of our algorithm is linear with respect to the number of input vertices; it takes around an hour for a model with around 10 thousand vertices.

6.2 Comparisons

Curved ODT. [Feng et al. 2018] is, to the best of our knowledge, the only existing algorithm designed to convert dense triangle meshes into coarse, curved approximations. The input and output are the same as in our algorithm. However, their method does not provide a bijective map between the input and output, does not guarantee to preserve features, has no bound on the distance to the input surface, and does not guarantee that the elements are positive. While our algorithm has been designed to process large collections of data automatically, exposing only a few intuitive options to control the faithfulness to the input, the reference implementation of the curved ODT method provided to us by the authors requires the user to choose multiple *per-model* parameters to achieve good results, and the parameters have a strong effect on the quality and validity of the result (as shown in [Feng et al. 2018, Figure 16]). We thus restricted our comparison to only a small selection of models (see additional material) that the authors of [Feng et al. 2018] processed for us.

From our discussions with the authors, we observed that curved ODT generates a *valid* output when we provide (1) a sufficiently large number of vertices and (2) a good local feature size sizing field (LFS) [Alliez et al. 2005] to efficiently spend the vertex budget in the regions with more geometrical details. Figure 21 shows an example of a model for which [Feng et al. 2018] fails to converge when using a uniform sizing field, while it succeeds when the sizing field is used.

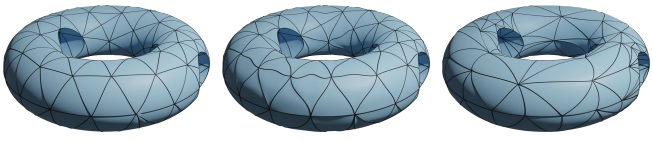


Fig. 22. Example of a BRep meshed with Gmsh where the optimization fails to untangle elements when fixing the surface. By allowing the surface to be modified, the mesh becomes "wiggly". Our method successfully generates a positive curved mesh.

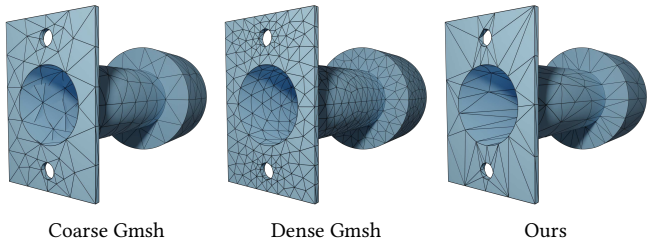


Fig. 23. Example of a STEP file meshed with Gmsh where, due to the low mesh density, the tetrahedralization is not positive. Gmsh manages to generate a positive mesh by using a denser initial tessellation. Since our method starts from a dense mesh and coarsen , it can successfully resolve the geometry.

In contrast, our algorithm can be run automatically on a large collection of geometrical models, it is guaranteed to have positive Jacobian (up to the use of floating-point predicates, Section 7), it preserves features, it automatically controls the density of the output depending on the desired user-provided distance threshold, and it provides a bijective map between the input mesh the boundary of the curved surface. For the model in Figure 21, our result contains 2037 elements, 18 times less than the curved ODT algorithm.

Gmsh. [Geuzaine and Remacle 2009] can only generate curved meshes from boundary representation (BRep), that is, the input is not exactly the same as ours. To compare both algorithms, we start from the BRep, and we generate a dense *linear* mesh that we use for our input. The Gmsh algorithm first constructs a curved mesh by fitting the high-order nodes to the BRep (possibly inverting elements) then performs mesh optimization to untangle them [Remacle et al. 2013]; thus has no guarantee to generate positive meshes while preserving the surface (Figure 22, left). Additionally, Gmsh algorithm cannot control the distance from the input when the untangling allows the surface to move, and thus the surface is "wiggly" and denser than our result (Figure 22, center). We also observed that if the initial surface mesh is not dense enough, Gmsh closes holes and cannot generate a valid tetrahedral mesh (Figure 23).

6.3 Flexibility

Since the input to our method is a triangle mesh, our method naturally supports a variety of input that can be easily converted into triangle meshes. For instance, $\mathcal{M}$ can be generated from marching an implicit surface or Catmull-Clark subdivision of a hand-made quad mesh (Figure 24). The bijective map ϕ^k is used, for instance, to transfer the geodesic distance field or color information from the input triangle mesh to the coarse curved mesh.

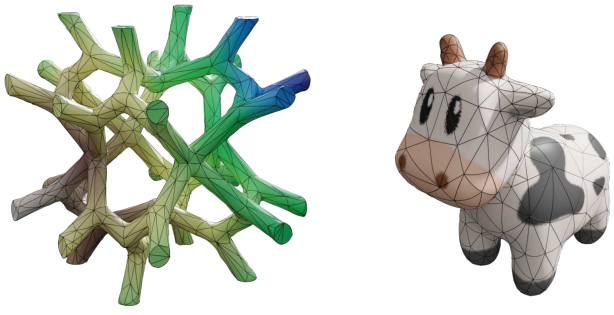


Fig. 24. Our algorithm processes triangle meshes that can be extracted from different formats: an implicit microstructure geometry from [Tozoni et al. 2020] or a subdivision surface from [Crane 2013]. The bijective map preserved on the surface allows taking advantage of the plethora of surface algorithms, including polyhedral geodesic computation [Mitchell et al. 1987] and texture mapping.

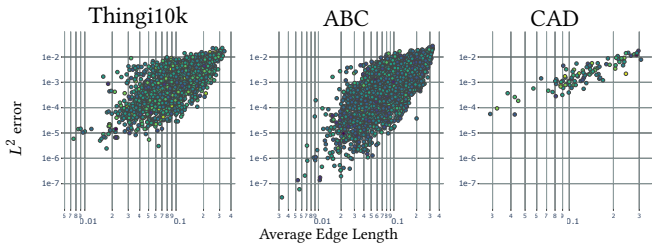


Fig. 25. L^2 error of the solution of the Poisson equation with respect to model size on our three datasets.

6.4 Applications

Large Scale Poisson. To show that our meshes are ready for simulation, we solve the Dirichlet problem for the Poisson equation

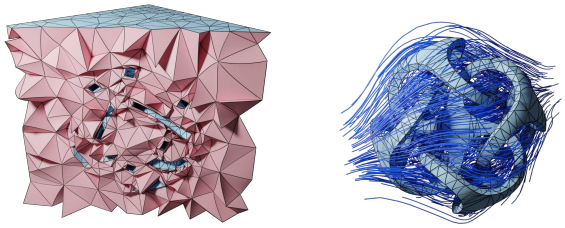
$$\Delta u = f, \quad u|_{\partial\Omega} = g,$$

where Ω is the domain (i.e., the mesh), f is the right-hand side, and g are the Dirichlet boundary conditions. To simplify the setup and the error measurements, we use fabricated solutions [Salari and Knupp 2000]. That is, we choose the function u_{exact} to be

$$\begin{aligned} u_{\text{exact}}(x_1, x_2, x_3) = \\ & 3/4\, e^{-((9x_1-2)^2+(9x_2-2)^2+(9x_3-2)^2)/4} + 3/4\, e^{-(9x_1+1)^2/49-(9x_2+1)/10-(9x_3+1)/10} \\ & + 1/2\, e^{-((9x_1-7)^2+(9x_2-3)^2+(9x_3-5)^2)/4} - 1/5\, e^{-(9x_1-4)^2-(9x_2-7)^2-(9x_3-5)^2}, \end{aligned}$$

then we plug it in the equation to obtain f (g is simply u_{exact}). Figure 25 shows the L^2 error (average) distribution across our three datasets using our quartic meshes with quadratic approximation of u (i.e., we use superparametric elements).

High Accuracy Fluids. Our curved meshes can be directly used to solve different partial differential equations (PDEs). For instance, by meshing the part outside the top shell and discarding the rest,



Tetrahedral mesh Curved simulation

Fig. 26. By meshing the region between a box and a complicated obstacle, we are able to perform non-linear fluid simulation on our curved mesh.

we can generate a curved background mesh for the Navier-Stokes equation (Figure 26).

Fast Animation. Our coarse curved meshes can be used as animation proxies as in [Mezger et al. 2007; Suwelack et al. 2013]. We first compute an as-coarse-as-possible curved mesh (i.e., we set ε to infinity). Then we apply the boundary condition to simulate an elastic distortion of the curved mesh using linear elements. Finally, we use our bijective map φ^4 to map the displacement back to the input high-detailed surface mesh (Figure 1). The results are almost indistinguishable to a classical pipeline (i.e., mesh the input mesh), but the runtime is 400 times faster (8s versus over 50 minutes).

7 LIMITATIONS AND CONCLUDING REMARKS

We introduce an automatic algorithm to convert dense triangle meshes into coarse, curved tetrahedral meshes whose boundary is within a user-controlled distance from the input mesh. Our algorithm supports feature preservation and generates meshes with bijective geometric maps and high quality, which are directly usable for FEM simulations.

Limitations. Our algorithm generates meshes with a C^0 geometric map. For most FEM applications, this is not an issue [Luo et al. 2002b; Xia and Qian 2017; Zaide et al. 2015]. However, for geometric modeling applications, where only the mesh boundary is used, the C^0 geometric map introduces normal discontinuities, which are undesirable. While the surface looks smooth from far away, plotting the reflection lines shows the discontinuity between the normals. We believe an exciting extension of our work would be to study the feasibility of using geometric maps that are C^1 [Lyche and Muntingh 2015] or C^2 [Lai and Schumaker 2007; Xia and Qian 2017]. A second limitation is that, in our implementation, the validity conditions (Definition 4.1) are currently checked using floating-point arithmetic, using heuristic numerical tolerances to account for rounding errors. While our implementation works on a large collection of models, it is possible to fail on others due to the inexact validity predicates. We are not aware of exact predicates for these conditions, and we believe that developing them is an interesting and challenging venue for future work.

Future Work. Our current high-order mesh optimization pipeline is preliminary, as it only supports vertex smoothing, collapse, and swap with simple validation criteria. We believe adding additional operations, allowing new nodes to exploit the curved geometric map, carefully analyze new energies for the high-order settings, and adapting the curved boundary could not only lead to a further increase in mesh quality and additional coarsening, but also better performance in the running time of the algorithms.

Conclusions. We believe that our work will foster the adoption of curved meshes, and open the door to a new family of geometry processing algorithms able to take advantage of this highly compact yet accurate shape representation.

ACKNOWLEDGMENTS

The authors would like to thank Leman Feng and Pierre Alliez for providing the source code of [Feng et al. 2018] and processing the models. 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 NSFAREER award 1652515, the NSFrants 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. The authors thank all the reviewers for their feedback.

REFERENCES

Remi Abgrall, Cécile Dobrzynski, and Algiane Froehly. 2012. *A method for computing curved 2D and 3D meshes via the linear elasticity analogy: preliminary results.* Research Report RR-8061. INRIA. 15 pages. https://hal.inria.fr/hal-00728850

R. Abgrall, C. Dobrzynski, and A. Froehly. 2014. A method for computing curved meshes via the linear elasticity analogy, application to fluid dynamics problems. *International Journal for Numerical Methods in Fluids* 76, 4 (2014), 246–266.

Noam Aigerman and Yaron Lipman. 2013. Injective and bounded distortion mappings in 3D. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.

Marc Alexa. 2019. Harmonic Triangulations. *ACM Trans. Graph.* 38, 4, Article 54 (July 2019), 14 pages. https://doi.org/10.1145/3306346.3322986

Marc Alexa. 2020. Conforming weighted delaunay triangulations. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–16.

Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. 2005. Variational tetrahedral meshing. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM.

Douglas N Arnold, Richard S Falk, and Ragnar Winther. 2000. Multigrid in H (div) and H (curl). *Numer. Math.* 85, 2 (2000), 197–217.

I. Babuska and B. Q. Guo. 1988. The h-p Version of the Finite Element Method for Domains with Curved Boundaries. *SIAM J. Numer. Anal.* 25, 4 (1988), 837–861. http://www.jstor.org/stable/2157607

I. Babuška and B.Q. Guo. 1992. The h, p and h-p version of the finite element method; basis theory and applications. *Advances in Engineering Software* 15, 3 (1992), 159 – 174. https://doi.org/10.1016/0965-9978(92)90097-Y

Adam W Bargteil and Elaine Cohen. 2014. Animation of deformable bodies with quadratic Bézier finite elements. *ACM Transactions on Graphics (TOG)* 33, 3 (2014), 27.

F. Bassi and S. Rebay. 1997. High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations. *J. Comput. Phys.* 138, 2 (1997), 251 – 285. https://doi.org/10.1006/jcph.1997.5454

Jean-Daniel Boissonnat and Steve Oudot. 2005. Provably good sampling and meshing of surfaces. *Graphical Models* 67, 5 (2005), 405–451.

D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silv a, M. Tarini, and D. Zorin. 2012. State of the Art in Quad Meshing. In *Eurographics STARS*.

John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. 2010. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–10.

David Cardoze, Alexandre Cunha, Gary L. Miller, Todd Phillips, and Noel Walkington. 2004. A Bézier-based Approach to Unstructured Moving Meshes. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) *(SCG '04)*. ACM, New York, NY, USA, 310–319. https://doi.org/10.1145/997817.997864

Siu-Wing Cheng, Tamal K Dey, Herbert Edelsbrunner, Michael A Facello, and Shang-Hua Teng. 2000. Silver exudation. *Journal of the ACM (JACM)* 47, 5 (2000), 883–904.

Siu-Wing Cheng, Tamal K Dey, and Joshua A Levine. 2008. A practical Delaunay meshing algorithm for a large class of domains. In *Proceedings of the 16th International Meshing Roundtable*. Springer, 477–494.

L Paul Chew. 1989. Constrained delaunay triangulations. *Algorithmica* 4, 1-4 (1989), 97–108.

L Paul Chew. 1993. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the ninth annual symposium on Computational geometry*. ACM, 274–280.

David Cohen-Steiner, Eric Colin De Verdiere, and Mariette Yvinec. 2002. Conforming Delaunay triangulations in 3D. In *Proceedings of the eighteenth annual symposium on Computational geometry*. ACM, 199–208.

Coreform. 2020. *Cubit*.

Keenan Crane. 2013. *Conformal geometry processing*. California Institute of Technology.

Sailkat Dey, Robert M O'bara, and Mark S Shephard. 1999. Curvilinear Mesh Generation in 3D.. In *IMR*. IMR, 407–417.

Veselin Dobrev, Patrick Knupp, Tzanio Kolev, Ketan Mittal, and Vladimir Tomov. 2019. The target-matrix optimization paradigm for high-order meshes. *SIAM Journal on Scientific Computing* 41, 1 (2019), B50–B68.

Cecile Dobrzynski and Ghina El Jannoun. 2017. *High order mesh untangling for complex curved geometries*. Research Report RR-9120. INRIA Bordeaux, équipe CARDAMOM. https://hal.inria.fr/hal-01632388

Qiang Du and Desheng Wang. 2003. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. *International journal for numerical methods in engineering* 56, 9 (2003), 1355–1373.

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

Luke Engvall and John A. Evans. 2016. Isogeometric triangular Bernstein–Bézier discretizations: Automatic mesh generation and geometrically exact finite element analysis. *Computer Methods in Applied Mechanics and Engineering* 304 (2016), 378 – 407. https://doi.org/10.1016/j.cma.2016.02.012

Luke Engvall and John A. Evans. 2017. Isogeometric unstructured tetrahedral and mixed-element Bernstein–Bézier discretizations. *Computer Methods in Applied Mechanics and Engineering* 319 (2017), 83 – 123. https://doi.org/10.1016/j.cma.2017.02.017

Luke Engvall and John A. Evans. 2018. Mesh Quality Metrics for Isogeometric Bernstein-Bézier Discretizations. *arXiv:1810.06975* (2018).

Darren Engwirda. 2016. Conforming restricted Delaunay mesh generation for piecewise smooth complexes. *CoRR* (2016). http://arxiv.org/abs/1606.01289

Patrick E Farrell, Matthew G Knepley, Lawrence Mitchell, and Florian Wechsung. 2019. PCPATCH: software for the topological construction of multigrid relaxation methods. *arXiv preprint arXiv:1912.08516* (2019).

Leman Feng, Pierre Alliez, Laurent Busé, Hervé Delingette, and Mathieu Desbrun. 2018. Curved optimal delaunay triangulation. *ACM Transactions on Graphics (TOG)* (2018).

Meire Fortunato and Per-Olof Persson. 2016. High-order unstructured curved mesh generation using the Winslow equations. *J. Comput. Phys.* 307 (2016), 1 – 14. https://doi.org/10.1016/j.jcp.2015.11.020

Xiao-Ming Fu, Yang Liu, and Baining Guo. 2015. Computing locally injective mappings by advanced MIPS. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. 2019. Feature Preserving Octree-Based Hexahedral Meshing. *Computer Graphics Forum* 38, 5 (2019), 135–149.

A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate. 2015. Optimization of a regularized distortion measure to generate curved high-order unstructured tetrahedral meshes. *Internat. J. Numer. Methods Engrg.* 103, 5 (2015), 342–363. https://doi.org/10.1002/nme.4888 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.4888

Abel Gargallo Peiró, Francisco Javier Roca Navarro, Jaume Peraire Guitart, and Josep Sarrate Ramos. 2013. High-order mesh generation on CAD geometries. In *Adaptive Modeling and Simulation 2013*. Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), 301–312.

P.L. George and H. Borouchaki. 2012. Construction of tetrahedral meshes of degree two. *Internat. J. Numer. Methods Engrg.* 90, 9 (2012), 1156–1182.

Christophe Geuzaine, Amaury Johnen, Jonathan Lambrechts, Jean-François Remacle, and Thomas Toulorge. 2015. *The Generation of Valid Curvilinear Meshes*. Springer International Publishing, Cham, 15–39. https://doi.org/10.1007/978-3-319-12886-3_2

Christophe Geuzaine and Jean-François Remacle. 2009. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331.

Arash Ghasemi, Lafayette K. Taylor, and James C. Newman, III. 2016. Massively Parallel Curved Spectral/Finite Element Mesh Generation of Industrial CAD Geometries in Two and Three Dimensions. *Fluids Engineering Division Summer Meeting* 50299 (2016). http://dx.doi.org/10.1115/FEDSM2016-7600

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

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.

Hao-Xiang Guo, Xiaohan Liu, Dong-Ming Yan, and Liu Yang. 2020. Cut-enhanced PolyCube-Maps for Feature-aware All-Hex Meshing. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 106:1–106:14.

S. Hahmann and G. . Bonneau. 2003. Polynomial surfaces interpolating arbitrary triangulations. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (2003), 99–109. https://doi.org/10.1109/TVCG.2003.1175100

Joos Heintz, Tomas Recio, and Marie-Françoise Roy. 1991. Algorithms in real algebraic geometry and applications to computational geometry. *Discrete and Computational Geometry: Papers from the DIMACS Special Year (JE Goodman, R. Pollack and W. Steiger, Eds.), AMS Press, Providence, RI* (1991), 137–163.

Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. 1994. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 295–302.

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

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TriWild: Robust Triangulation with Curve Constraints. *ACM Trans. Graph.* 38, 4, Article 52 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3323011

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 39, 4, Article 117 (July 2020), 18 pages. https://doi.org/10.1145/3386569.3392385

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.

Alec Jacobson, Daniele Panozzo, C Schüller, O Diamanti, Q Zhou, N Pietroni, et al. 2016. *libigl: A simple C++ geometry processing library, 2016*.

A. Jameson, J. Alonso, and M. McMullen. 2002. Application of a non-linear frequency domain solver to the Euler and Navier-Stokes equations. In *40th AIAA Aerospace Sciences Meeting & Exhibit*.

Clément Jamin, Pierre Alliez, Mariette Yvinec, and Jean-Daniel Boissonnat. 2015. CGALmesh: a generic framework for delaunay mesh generation. *ACM Transactions on Mathematical Software (TOMS)* 41, 4 (2015), 23.

Zhongshi Jiang, Teseo Schneider, Denis Zorin, and Daniele Panozzo. 2020. Bijective projection in a shell. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–18.

Xiangmin Jiao and Duo Wang. 2012. Reconstructing high-order surfaces for meshing. *Engineering with Computers* 28, 4 (2012), 361–373.

Amaury Johnen, J-F Remacle, and Christophe Geuzaine. 2013. Geometrical validity of curvilinear finite elements. *J. Comput. Phys.* 233 (2013), 359–372.

Patrick M. Knupp. 2000. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part I, A framework for surface mesh optimization. *Internat. J. Numer. Methods Engrg.* 48, 3 (2000), 401–420.

Patrick M. Knupp. 2002. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II, A framework for volume mesh optimization and the condition number of the Jacobian matrix. *Internat. J. Numer. Methods Engrg.* 48, 8 (2002), 1165–1185.

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)*.

Venkat Krishnamurthy and Marc Levoy. 1996. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 313–324.

Ming-Jun Lai and Larry L Schumaker. 2007. *Spline functions on triangulations*. Vol. 110. Cambridge University Press.

Bruno Lévy. 2015. *Geogram*.

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-Hex Meshing Using Singularity-Restricted Field. *ACM Trans. Graph.* 31, 6, Article 177 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366196

Hongwei Lin, Wei Chen, and Hujun Bao. 2007. Adaptive patch-based mesh fitting for reverse engineering. *Computer-Aided Design* 39, 12 (2007), 1134 – 1142. https://doi.org/10.1016/j.cad.2007.10.002

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.

Qiukai Lu, Mark S. Shephard, Saurabh Tendulkar, and Mark W. Beall. 2013. Parallel Curved Mesh Adaptation for Large Scale High-Order Finite Element Simulations. In *Proceedings of the 21st International Meshing Roundtable*, Xiangmin Jiao and Jean-Christophe Weill (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 419–436.

Qiukai Lu, Mark S Shephard, Saurabh Tendulkar, and Mark W Beall. 2014. Parallel mesh adaptation for high-order finite element methods with curved element geometry. *Engineering with Computers* 30, 2 (2014), 271–286.

Xiaojuan Luo, Mark S Shephard, Lie-Quan Lee, Cho Ng, and Lixin Ge. 2008. Tracking adaptive moving mesh refinements in 3d curved domains for large-scale higher order finite element simulations. In *Proceedings of the 17th International Meshing Roundtable*. Springer, 585–601.

Xiaojuan Luo, Mark S Shephard, and Jean-Francois Remacle. 2001. The influence of geometric approximation on the accuracy of high order methods. *Rensselaer SCOREC*

report 1 (2001).

Xiaojuan Luo, Mark S. Shephard, Jean-François Remacle, Robert M. O'Bara, Mark W. Beall, Barna A. Szabó, and Ricardo Actis. 2002a. p-Version Mesh Generation Issues. In *IMR*.

Xiaojuan Luo, Mark S Shephard, Jean-François Remacle, Robert M O'Bara, Mark W Beall, Barna A Szabó, and Ricardo Actis. 2002b. p-Version Mesh Generation Issues.. In *IMR*. 343–354.

Tom Lyche and Georg Muntingh. 2015. Simplex Spline Bases on the Powell-Sabin 12-Split: Part I. *arXiv preprint arXiv:1505.01798* (2015).

Julian Marcon, Joaquim Peiró, David Moxey, Nico Bergemann, Henry Bucklow, and Mark R Gammon. 2019. A semi-structured approach to curvilinear mesh generation around streamlined bodies. In *AIAA Scitech 2019 Forum*. 1725.

Zoë Marschner, David Palmer, Paul Zhang, and Justin Solomon. 2020. Hexahedral Mesh Repair via Sum-of-Squares Relaxation. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 133–147.

Johannes Mezger, Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2007. A Finite Element Method for Interactive Physically Based Shape Modelling with Quadratic Tetrahedra. (2007).

Johannes Mezger, Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straśer. 2009. Interactive physically-based shape editing. *Computer Aided Geometric Design* 26, 6 (2009), 680 – 694. https://doi.org/10.1016/j.cagd.2008.09.009 Solid and Physical Modeling 2008.

Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.

D. Moxey, D. Ekelschot, Ü. Keskin, S.J. Sherwin, and J. Peiró. 2016. High-order curvilinear meshing using a thermo-elastic analogy. *Computer-Aided Design* 72 (2016), 130 – 139. https://doi.org/10.1016/j.cad.2015.09.007 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

D. Moxey, M.D. Green, S.J. Sherwin, and J. Peiró. 2015. An isoparametric approach to high-order curvilinear boundary-layer meshing. *Computer Methods in Applied Mechanics and Engineering* 283 (2015), 636 – 650. https://doi.org/10.1016/j.cma.2014.09.019

Dave Moxey, Michael Turner, Julian Marcon, and Joaquim Peiro. 2018. Nekmesh: An open-source high-order mesh generator. (2018).

Michael Murphy, David M Mount, and Carl W Gable. 2001. A point-placement strategy for conforming Delaunay tetrahedralization. *International Journal of Computational Geometry & Applications* 11, 06 (2001), 669–682.

J.Tinsley Oden. 1994. Optimal h-p finite element methods. *Computer Methods in Applied Mechanics and Engineering* 112, 1 (1994), 309 – 331. https://doi.org/10.1016/0045-7825(94)90032-9

David Palmer, David Bommes, and Justin Solomon. 2020. Algebraic Representations for Volumetric Frame Fields. *ACM Trans. Graph.* 39, 2, Article 16 (April 2020), 17 pages. https://doi.org/10.1145/3366786

Abel Gargallo Peiró, Eloi Ruiz Gironés, Francisco J Navarro, and Josep Sarrate Ramos. 2015. On curving high-order hexahedral meshes.

Abel Gargallo Peiró, Xevi Roca, Jaime Peraire, and Josep Sarrate. 2014. Defining Quality Measures for Validation and Generation of High-Order Tetrahedral Meshes. In *Proceedings of the 22nd International Meshing Roundtable*, Josep Sarrate and Matthew Staten (Eds.). Springer International Publishing, Cham, 109–126.

Joaquim Peiró, Spencer J. Sherwin, and Sergio Giordana. 2008. Automatic reconstruction of a patient-specific high-order surface representation and its application to mesh generation for CFD calculations. *Medical & Biological Engineering & Computing* 46, 11 (01 Nov 2008), 1069–1083. https://doi.org/10.1007/s11517-008-0390-3

Per-Olof Persson and Jaime Peraire. 2009. Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. https://arc.aiaa.org/doi/10.2514/6.2016-3178

Pointwise. 2018. *High Order Mesh Generation at Pointwise*. Accessed: 2018-11-14.

Roman Poya, Ruben Sevilla, and Antonio J. Gil. 2016. A unified approach for a posteriori high-order curved mesh generation using solid mechanics. *Computational Mechanics* 58, 3 (01 Sep 2016), 457–490. https://doi.org/10.1007/s00466-016-1302-2

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2, Article 16 (April 2017), 16 pages. https://doi.org/10.1145/2983621

Jean-François Remacle, Thomas Toulorge, and Jonathan Lambrechts. 2013. Robust untangling of curvilinear meshes. In *Proceedings of the 21st International meshing roundtable*. Springer, 71–83.

Xevi Roca, Abel Gargallo-Peiró, and Josep Sarrate. 2012. Defining Quality Measures for High-Order Planar Triangles and Curved Mesh Generation. In *Proceedings of the 20th International Meshing Roundtable*, William Roshan Quadros (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 365–383.

Eloi Ruiz-Gironés, Abel Gargallo-Peiró, Josep Sarrate, and Xevi Roca. 2017. An augmented Lagrangian formulation to impose boundary conditions for distortion based mesh moving and curving. *Procedia Engineering* 203 (2017), 362 – 374. https://doi.org/10.1016/j.proeng.2017.09.820 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

Eloi Ruiz-Gironés, Xevi Roca, and Jose Sarrate. 2016a. High-order mesh curving by distortion minimization with boundary nodes free to slide on a 3D CAD representation. *Computer-Aided Design* 72 (2016), 52 – 64. https://doi.org/10.1016/j.cad.2015.06.011 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

Eloi Ruiz-Gironés, Josep Sarrate, and Xevi Roca. 2016b. Generation of Curved High-order Meshes with Optimal Quality and Geometric Accuracy. *Procedia Engineering* 163 (2016), 315 – 327. https://doi.org/10.1016/j.proeng.2016.11.108 25th International Meshing Roundtable.

Jim Ruppert. 1995. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of algorithms* 18, 3 (1995), 548–585.

Kambiz Salari and Patrick Knupp. 2000. *Code Verification by the Method of Manufactured Solutions*. Technical Report. https://doi.org/10.2172/759450

N Schlömer. 2020. nschloe/meshio: Input/output for many mesh formats. *Zenodo. doi* 10 (2020).

Mark S. Shephard, Joseph E. Flaherty, Kenneth E. Jansen, Xiangrong Li, Xiaojuan Luo, Nicolas Chevaugeon, Jean-François Remacle, Mark W. Beall, and Robert M. O'Bara. 2005. Adaptive mesh generation for curved domains. *Applied Numerical Mathematics* 52, 2 (2005), 251 – 271. https://doi.org/10.1016/j.apnum.2004.08.040 ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation.

SJ Sherwin and J Peiró. 2002. Mesh generation in curvilinear domains using high-order elements. *Internat. J. Numer. Methods Engrg.* 53, 1 (2002), 207–223.

Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18, 3 (1997), 305–363.

Jonathan Richard Shewchuk. 1998. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the fourteenth annual symposium on Computational geometry*. ACM, 86–95.

Jonathan Richard Shewchuk. 2002. Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery.. In *IMR*. 193–204.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2, Article 11 (Feb. 2015), 36 pages.

Hang Si and Klaus Gärtner. 2005. Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations.. In *IMR*. Springer, 147–163.

Hang Si and Jonathan Richard Shewchuk. 2014. Incrementally constructing and updating constrained Delaunay tetrahedralizations with finite-precision coordinates. *Eng. Comput. (Lond.)* 30, 2 (2014), 253–269.

C. Sorger, Felix Frischmann, Stefan Kollmannsberger, and Ernst Rank. 2014. TUM.GeoFrame: automated high-order hexahedral mesh generation for shell-like structures. *Eng. Comput.* 30, 1 (2014), 41–56. https://doi.org/10.1007/s00366-012-0284-8

Mike Stees and Suzanne M. Shontz. 2017. A high-order log barrier-based mesh generation and warping method. *Procedia Engineering* 203 (2017), 180 – 192. https://doi.org/10.1016/j.proeng.2017.09.806 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

Ryan S. Glasby Steve L. Karman, J T. Erwin and Douglas Stefanski. 2016. High-Order Mesh Curving Using WCN Mesh Optimization. In *46th AIAA Fluid Dynamics Conference, AIAA AVIATION Forum*. https://arc.aiaa.org/doi/10.2514/6.2016-3178

Stefan Suwelack, Dimitar Lukarski, Vincent Heuveline, Rüdiger Dillmann, and Stefanie Speidel. 2013. Accurate Surface Embedding for Higher Order Finite Elements. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Anaheim, California) *(SCA'13)*. ACM, New York, NY, USA, 187–192. https://doi.org/10.1145/2485895.2485914

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. 2003. Optimized spatial hashing for collision detection of deformable objects.. In *Vmv*, Vol. 3. 47–54.

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

Wei-hua Tong and Tae-wan Kim. 2009. High-order approximation of implicit surfaces by G1 triangular spline surfaces. *Computer-Aided Design* 41, 6 (2009), 441–455.

Thomas Toulorge, Christophe Geuzaine, Jean-François Remacle, and Jonathan Lambrechts. 2013. Robust untangling of curvilinear meshes. *J. Comput. Phys.* 254 (2013), 8 – 26. https://doi.org/10.1016/j.jcp.2013.07.022

Thomas Toulorge, Jonathan Lambrechts, and Jean-François Remacle. 2016. Optimizing the geometrical accuracy of curvilinear meshes. *J. Comput. Phys.* 310 (2016), 361 – 380. https://doi.org/10.1016/j.jcp.2016.01.023

Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun. 2009. Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Transactions on Graphics* 28, 3 (2009), Art–No.

Davi Colli Tozoni, Jérémie Dumas, Zhongshi Jiang, Julian Panetta, Daniele Panozzo, and Denis Zorin. 2020. A low-parametric rhombic microstructure family for irregular lattices. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 101–1.

Michael Turner. 2017. *High-order mesh generation for CFD solvers*. Ph.D. Dissertation. Imperial College London.

Michael Turner, Joaquim Peiró, and David Moxey. 2016. A Variational Framework for High-order Mesh Generation. *Procedia Engineering* 163 (2016), 340 – 352. https:

//doi.org/10.1016/j.proeng.2016.11.069 25th International Meshing Roundtable.
S Pratap Vanka. 1986. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *J. Comput. Phys.* 65, 1 (1986), 138–158.
Ruimin Wang, Ligang Liu, Zhouwang Yang, Kang Wang, Wen Shan, Jiansong Deng, and Falai Chen. 2016. Construction of manifolds via compatible sparse representations. *ACM Transactions on Graphics (TOG)* 35, 2 (2016), 1–10.
Songtao Xia and Xiaoping Qian. 2017. Isogeometric analysis with Bézier tetrahedra. *Computer Methods in Applied Mechanics and Engineering* 316 (2017), 782 – 816. https://doi.org/10.1016/j.cma.2016.09.045 Special Issue on Isogeometric Analysis: Progress and Challenges.
Zhong Q. Xie, Ruben Sevilla, Oubay Hassan, and Kenneth Morgan. 2013. The generation of arbitrary order curved meshes for 3D finite element analysis. *Computational Mechanics* 51, 3 (01 Mar 2013), 361–374. https://doi.org/10.1007/s00466-012-0736-4
Yuxuan Yu, Xiaodong Wei, Angran Li, Jialei Ginny Liu, Jeffrey He, and Yongjie Jessica Zhang. 2020. HexGen and Hex2Spline: Polycube-based Hexahedral Mesh Generation and Spline Modeling for Isogeometric Analysis Applications in LS-DYNA. (2020). arXiv:2011.14213 [cs.CG]
Alex Yvart, Stefanie Hahmann, and Georges-Pierre Bonneau. 2005a. Hierarchical Triangular Splines. *ACM Trans. Graph.* 24, 4 (Oct. 2005), 1374–1391. https://doi.org/10.1145/1095878.1095885
Alex Yvart, Stefanie Hahmann, and G-P Bonneau. 2005b. Smooth adaptive fitting of 3D models using hierarchical triangular splines. In *International Conference on Shape Modeling and Applications 2005 (SMI'05)*. IEEE, 13–22.
Daniel W Zaide, Qiukai Lu, and Mark S Shephard. 2015. A comparison of C0 and G1 continuous curved tetrahedral meshes for high-order finite element simulations. *Proc. 24th International Meshing Roundtable. Elsevier, New York* (2015).
Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. 2020. Octahedral Frames for Feature-Aligned Cross Fields. *ACM Trans. Graph.* 39, 3, Article 25 (April 2020), 13 pages. https://doi.org/10.1145/3374209
S. Zhang, Z. Li, H. Zhang, and J. Yong. 2011. Multi-resolution Mesh Fitting by B-spline Surfaces for Reverse Engineering. In *2011 12th International Conference on Computer-Aided Design and Computer Graphics*. 251–257. https://doi.org/10.1109/CAD/Graphics.2011.65
Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).
V.S. Ziel, H. Bériot, O. Atak, and G. Gabard. 2017. Comparison of 2D boundary curving methods with modal shape functions and a piecewise linear target mesh. *Procedia Engineering* 203 (2017), 91 – 101. https://doi.org/10.1016/j.proeng.2017.09.791 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.
Denis Zorin. 2000. Subdivision for modeling and animation. *SIGGRAPH2000 course notes* (2000).

A THE GEOMETRIC MAP IS BIJECTIVE

Consider a connected 3-dimensional compact manifold (curved) tetrahedral mesh $\mathcal{M} = \{\sigma_i\}, i = 1, \ldots, n$ with $\sigma_i = g_i(\hat{\tau})$, $\hat{\tau}$ a regular unit tetrahedron, $\det(J_{g_i}) > 0$ at all points including the boundary, and σ_i and σ_j agree on a shared face. Let $\mathcal{M}_D$ be the domain obtained by copies of $\hat{\tau}$ and identifying $\hat{\tau}_i$ along common faces. We then define the map

$$\sigma \colon M_D \to \mathbb{R}^3,$$

by setting $\sigma|_{\hat{\tau}_i} = \sigma_i$.

Proposition A.1. *Suppose $\sigma|_{\partial\mathcal{M}_D}$ is injective. Then σ is injective on the whole domain $\mathcal{M}_D$.*

Proof. Our argument closely follow from [Aigerman and Lipman 2013, Appendix B] and [Lipman 2014, Theorem 1].

We consider point $y \in \mathbb{R}^3$ in general position, but not in the faces, edges, vertices, or any plane spanned by a linear face of $\mathcal{M}$. For each tetrahedron τ, we construct the map $\hat{\Psi}$ as a composition of $\sigma|_{\partial\hat{\tau}_i}$ (restricted to the triangular faces of the regular tetrahedron) and the projection map χ to the unit sphere centered around y.

For each triangular face δ of $\partial\hat{\tau}_i$, we parametrize the image of $\sigma|_\delta$ as $x(u, v)$ (note that since $\det J_{g_i} > 0$, the image is a non-degenerate surface homeomorphic to a disk). Construct the normal field $n(u, v) = \frac{\partial}{\partial u}x \times \frac{\partial}{\partial v}x$ and note that $x(u, v), n(u, v)$ are polynomial functions. The algebraic curve $n(u, v) \cdot (x(u, v) - y) = 0$ partitions the surface into finite number of patches, where on each patch the orientation of $\hat{\Psi}$ is constant. We can further triangulate such semi-algebraic sets [Heintz et al. 1991, §5.7].

Similar to [Aigerman and Lipman 2013, Appendix B], we count the number of pre-images of y, which equals to the degree in general positions,

$$\deg(\sigma)(y) = \sum_{i=1}^{n} \deg(\hat{\Psi}|_{\tau_i}) = \deg(\hat{\Psi}|_{\partial\mathcal{M}}).$$

Since $\sigma|_{\partial\mathcal{M}_D}$ is injective, and furthermore

$$\deg(\hat{\Psi}|_{\partial\mathcal{M}}) = \deg\chi|_{\partial\mathcal{M}} = \begin{cases} 1, & y \in \mathcal{M} \\ 0, & y \notin \mathcal{M}. \end{cases}$$

Thus we have shown the map is injective for the general positions for y, and it remains to be shown that the map is an open map, which follows from the same argument of [Lipman 2014, Lemma 2]. □

B LOCAL OPERATIONS

[Jiang et al. 2020, Fig. 11] introduces a set of *valid* local operations to modify the shell, including edge split, edge collapse, edge flip and vertex smoothing. The operations are an revanalog of the triangle mesh edit operations [Dunyach et al. 2013], by simultaneously editing the shared connectivity of the bottom, middle, and top surface of the shell. [Jiang et al. 2020, Theorem 3.7] outlines invariant conditions, which maintains the shell projection to be bijective.

Our algorithm follows and extends the local operations therein to the high order setting. In addition to the existing conditions, we also validate the curved volumetric mesh in the shell. The algorithm maintains the global intersection free bottom (top) surface with a dynamic hash grid [Teschner et al. 2003]. Then for each prism, we check the positivity (defined by the determinant of Jacobian of the geometric map) of the prismatic element (each decomposed into three tetrahedra).

In the presence of feature annotation and feature straightening (Section 5), more care is taken to maintain the valid correspondence between the grouped feature chains and the curved edges: edge flip is disabled on the edges annotated as features; collapse is only allowed when it does not degenerate the chain and the two endpoints of the edge are on the same chain. Since we require a map from the original input edges, we insert additional degrees of freedom in the input mesh. When performing edge split, the new inserted vertex is chosen from the existing vertices from the input, which lie in the pre-image of the current edge. For vertex smoothing (more specifically pan), the target location is again limited to the set of input vertices.

C BOUNDARY PRESERVING TETGEN COMPARISON

We compared our conforming tetrahedral meshing algorithm (Section 4.3) with TetGen on the linear shells (triangle meshes) of 3522 models from the Thingi10k dataset, giving each model sufficient computing resources (2 hours maximum running time and 32GB memory usage). Inheriting the robustness from TetWild, our method

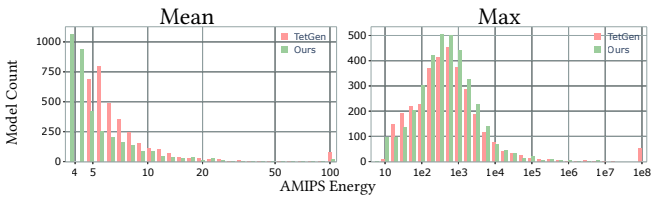


Fig. 27. Histogram of the mean and maximum conformal AMIPS energy [Rabinovich et al. 2017] of the output of our method and TetGen.

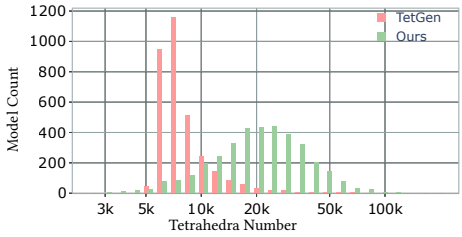


Fig. 28. Histogram of output tetrahedra number for TetGen and our method.

successfully processed all the inputs while preserving the triangulation, while TetGen fails on 224 models (215 models are not conforming, and 9 models have no output).

In Figure 27, we show the average and maximum element quality of the output of our method and TetGen. Our method has a better average and maximum output quality than TetGen. Note that in the quality plot, the "tail" of TetGen's distribution is longer than ours. The maximum average energy of TetGen's output and ours are 3×10^8 and 3×10^5 respectively. The largest maximum energy of TetGen's output and ours are 3×10^{12} and 7×10^6 respectively. Our method generates denser output (Figure 28), but our focus is on robustness instead of efficiency in this step.