Tetrahedral Meshing in the Wild

YIXIN HU, New York University
QINGNAN ZHOU, Adobe Research
XIFENG GAO, New York University
ALEC JACOBSON, University of Toronto
DENIS ZORIN, New York University
DANIELE PANOZZO, New York University

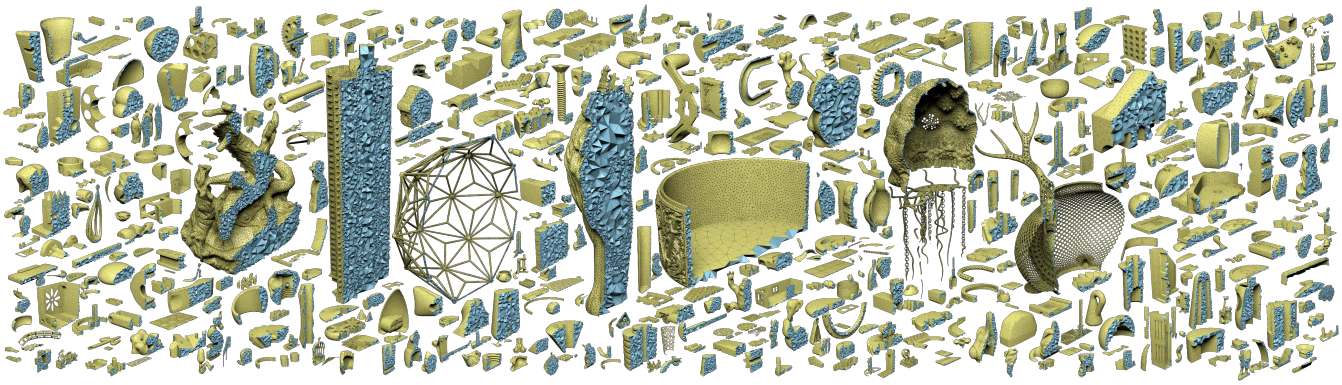


Fig. 1. A selection of the ten thousand meshes *in the wild* tetrahedralized by our novel tetrahedral meshing technique.

We propose a novel tetrahedral meshing technique that is unconditionally robust, requires no user interaction, and can directly convert a triangle soup into an analysis-ready volumetric mesh. The approach is based on several core principles: (1) initial mesh construction based on a fully robust, yet efficient, filtered exact computation (2) explicit (automatic or user-defined) tolerancing of the mesh relative to the surface input (3) iterative mesh improvement with guarantees, at every step, of the output validity. The quality of the resulting mesh is a direct function of the target mesh size and allowed tolerance: increasing allowed deviation from the initial mesh and decreasing the target edge length both lead to higher mesh quality.

Our approach enables "black-box" analysis, i.e. it allows to automatically solve partial differential equations on geometrical models available in the wild, offering a robustness and reliability comparable to, e.g., image processing algorithms, opening the door to automatic, large scale processing of real-world geometric data.

CCS Concepts: • **Mathematics of computing → Mesh generation**;

Additional Key Words and Phrases: Mesh Generation, Tetrahedral Meshing, Robust Geometry Processing

Authors' addresses: Yixin Hu, New York University, yh1998@nyu.edu; Qingnan Zhou, Adobe Research, qzhou@adobe.com; Xifeng Gao, New York University, gxf.xisha@gmail.com; Alec Jacobson, University of Toronto, jacobson@cs.toronto.edu; Denis Zorin, New York University, dzorin@cs.nyu.edu; Daniele Panozzo, New York University, panozzo@nyu.edu.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.
© 2018 Association for Computing Machinery.
0730-0301/2018/8-ART60 $15.00
https://doi.org/10.1145/3197517.3201353

ACM Reference Format:
Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild . *ACM Trans. Graph.* 37, 4, Article 60 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

1 INTRODUCTION

Triangulating the interior of a shape is a fundamental subroutine in 2D and 3D geometric computation.

For two-dimensional problems requiring meshing a domain, robust and efficient software for constrained Delaunay triangulation problem has been a tremendous boon to the development of robust and efficient automatic computational pipelines, in particular ones requiring solving PDEs. Robust 2D triangulations inside a given polygon boundary are also an essential spatial partitioning useful for fast point location, path traversal, and distance queries.

In 3D, the problem of robustly triangulating the interior of a given triangle surface mesh is just as well, if not more, motivated. While tremendous progress was made on various instances of the problem, it is far from solved by existing methods. While pipelines involving 3D *tetrahedralization* of smooth implicit surfaces are quite mature, pipelines using meshes as input either are limited to simple shapes or routinely fallback on manual intervention. The user may have to "fix" input surface meshes to cajole meshers to succeed due to unspoken pre-conditions, or output tetrahedral meshes must be repaired due to failure to meet basic post-conditions (such as manifoldness). Existing methods typically fail too often to support automatic pipelines, such as massive data processing for machine learning applications, or shape optimization. In many cases, while meshing may succeed, the size of the output mesh may be

prohibitively expensive for many applications, because a method lacks control between the quality of approximation of the input surface and the size of the output mesh. Even when such controls are present, hard-to-detect features of the input mesh may not be preserved.

In this paper, we propose a new approach to mesh domains that are represented (often ambiguously) by arbitrary meshes, with no assumptions on mesh manifoldness, watertightness, absence of self-intersections etc. Rather than viewing mesh repair as a separate preprocessing problem, we recognize the fact, that "clean" meshes are more of an exception than a rule in many settings.

The key features of our approach, based on careful analysis of practical meshing problems, and shortcomings of existing state of the art solutions are:

- We consider the input as fundamentally imprecise, allowing deviations from the input within user-defined envelope of size ϵ;
- We make no assumptions about the input mesh structure, and reformulate the meshing problem accordingly;
- We follow the principle that robustness comes first (i.e., the algorithm should produce a valid and, to the extent possible, useful output for a maximally broad range of inputs), with quality improvement done to the extent robustness constraints allow.
- While allowing deviations from the input, which is critical both for quality and performance, we aim to make our algorithm conservative, using the input surface mesh as a starting point for 3D mesh construction, rather than discarding its connectivity and using surface sampling only.

Our method is explicitly designed to output floating point coordinates, but at the same time is strictly *closed* under rationals allowing it to fit neatly into robust, exact rational computational geometry pipelines.

We empirically compare both the performance and robustness of state-of-the-art methods and our novel method on a large database of 10 thousand models from the web [Zhou and Jacobson 2016]. To foster replicability of results, we release a complete reference implementation of our algorithm, all the data shown in the paper, and scripts to reproduce our results.

Our method –while slower– demonstrates a significant improvement in robustness and quality of the results on a number of quality measures, when applied to meshes found *in the wild*.

2 RELATED WORK

Tetrahedral mesh generation has remained a perennial problem, both for computational geometers and practitioners in graphics, physics and engineering ([Carey 1997; Cheng et al. 2012; Owen 1998]). We are specifically interested in methods that are *constrained* to output a 3D tetrahedral mesh whose 2D surface closely matches an input surface. We categorize related work with respect to the high-level methodology employed. We place special emphasis on methods with reproducible results thanks to their openly accessible implementations. One confusion during comparisons is that most existing software implements *multiple* algorithms, triggered discretely (and somewhat discreetly) by input flags or parameters (e.g., TetGen or CGAL). Our comparisons are done in best faith and using default parameters where applicable; when controls similar to the ones used in our method are available, we tried to choose them in a similar way.

Background Grids. In 3D, a regular lattice of points is trivial to tetrahedralize (e.g., using either five, six, or 12 tetrahedra per cube). To tetrahedralize the interior of a solid given its surface, *grid-based* methods fill the ambient space with either a uniform grid or an adaptive octree. Grid cells far from the surface can be tetrahedralized immediately and efficiently using a predefined, combinatorial stencil, with excellent quality. Trouble arises for boundary cells.

Molino et al. [2003] propose the red-green tetrahedron refinement strategy, while cells intersecting the domain boundary are pushed into the domain via physics-inspired simulation. Alternatively, boundary cells can be cut into smaller pieces [Bronson et al. 2012]. Labelle & Shewchuck [2007] snap vertices to the input surface and cut crossing elements. This method provides bounds on dihedral angles and a proof of convergence for sufficiently *smooth* (bounded curvature) isosurface input. Doran et al. [2013] improves this method to detect and handle feature curves, providing an open source implementation, Quartet [Bridson and Doran 2014] with which we thoroughly compare. *Average* element quality tends to be good: for volumes with high volume-to-surface ratio, most of the mesh will be filled by the high-quality stencil. Near the boundary, grid-based methods struggle to simultaneously provide parsimony and element quality: either the surface is far denser than the interior making volume gradation difficult to control or the surface is riddled with low-quality elements.

Delaunay. The problem of tetrahedralizing a *set of points* is very well studied [Cheng et al. 2012; Sheehy 2012]. Efficient, scalable [Remacle 2017] algorithms exist to create Delaunay meshes.

When the input includes surface mesh constraints, the challenge is to extend the notion of a Delaunay mesh in a meaningful way. In two dimensions, constrained Delaunay methods provide a satisfactory solution. In contrast to 2D, the situation in 3D is immediately complicated by the fact that there exist polyhedra that cannot be tetrahedralized without adding extra interior Steiner vertices [Schönhardt 1928].

The simple and elegant idea of *Delaunay refinement* [Chew 1993; Ruppert 1995; Shewchuk 1998] is to insert new vertices at the center of the circumscribed sphere of the worst tetrahedron measured by radius-to-edge ratio. This approach guarantees termination and provides bounds on radius-edge ratio. This approach has been robustly implemented by many [Jamin et al. 2015; Si 2015], and, in our experiments, proved to be consistently successful. However, robustness problems immediately appear if the boundary facets have to be preserved.

More importantly, even in situations when the method is guaranteed to produce a mesh with bounded radius-to-edge ratio, it does not –unlike the 2D case– guarantee that quality measures relevant for applications are sufficiently good. The notorious "sliver" tetrahedra satisfy the radius-to-edge ratio criteria. Thus, unavoidably, Delaunay refinement needs to be followed by various mesh improvement heuristics: exudation [Cheng et al. 2000], Lloyd relaxation [Du and Wang 2003], ODT relaxation [Alliez et al. 2005],

or vertex perturbation [Tournois et al. 2009]. Our approach also relies on a variational-type mesh improvement (Section 3.2). *Conforming Delaunay tetrahedralization* [Cohen-Steiner et al. 2002; Murphy et al. 2001] splits input boundary by inserting additional Steiner points, until all input faces appear as supersets of element faces. Even with additional assumptions on the input, this process may require impractically many additional points and tetrahedra. In contrast, *constrained Delaunay tetrahedralization* [Chew 1989; Shewchuk 2002a; Si and Gärtner 2005; Si and Shewchuk 2014] proposes to relax the Delaunay requirement for boundary faces so fewer Steiner points are needed. The popular open source software TetGen [Si 2015] is based on constrained Delaunay tetrahedralization, enforcing inclusion of input faces in the mesh.

Restricted Delaunay tetrahedralization [Boissonnat and Oudot 2005; Cheng et al. 2008] completely resamples the input surface to obtain better tet quality while generating a good approximation of the domain boundary at the same time. The software DelPSC and CGAL 3D tetrahedral meshing module [Dey and Levine 2008; Jamin et al. 2015] is based on this approach. Engwirda [2016] uses an advancing front method as a refinement and point placement strategy for constructing a restricted Delaunay mesh.

Variations of these methods are difficult to implement robustly, as in their original form they require exact predicates that go beyond the typically available set, so a careful reduction to the robustly implementable operations is needed. This may account for a percentage of failures that we observe.

A conceptual feature of many restricted Delaunay meshers (using meshes as input) is that they do not allow any slack on the boundary geometry, thus requiring heavy refinement in certain cases to achieve acceptable quality, for any target tetrahedron size. However, tetrahedra incident at features are invariably excluded from quality improvement.

In contrast, our algorithm by design, admits practical robust implementation, and, also by design, allows the surface to change within user-specified bounds, which greatly reduces unnecessary over-refinement due to surface irregularities.

The state-of-the-art method based on restricted Delaunay refinement, [Jamin et al. 2015], is highly robust for important classes of inputs (smooth implicit surfaces) and yields high-quality meshes. However, as we demonstrate in the results section, if the input is polygonal, it cannot be easily reduced to the problem of meshing an implicit surface, due to nonsmoothness, and the need for feature preservation. Currently, [Jamin et al. 2015] and related methods preserve features using the protection ball method: spheres are placed on feature points and weighted Delaunay meshing and refinement are performed, treating ball radii as point weights. This approach requires explicit detection and representation of feature lines; in its current form, it results in reduction of robustness and in some cases over refinement.

Variational meshing. The duality between Delaunay meshes and Voronoi diagrams, leads to a *variational* or energy-minimizing view of the meshing problem. Centroidal Voronoi Tessellation energy minimizers can leverage Lloyd's algorithm of BFGS optimization to produce regular or adaptive meshes with well spaced vertices [Du and Wang 2003], though this does not guarantee good element quality [Eppstein 2001]. An alternative is to minimize the "Optimal Delaunay Triangulation" energy [Alliez et al. 2005; Chen and Xu 2004], for better element quality. These algorithms require an initial starting point (which cannot be generated starting from noisy input geometry), in order to stay near any input surface constraints. Our method is designed to generate this valid starting point, and it then uses a variant of these methods, which is designed to work with a hybrid kernel, to improve quality.

Other variational mesh improvement methods exist [Gargallo-Peiró et al. 2013; Klingner and Shewchuk 2007; Misztal and Bærentzen 2012], but all require and depend heavily on the initial base mesh. In contrast, we propose a complete meshing algorithm. Our first step generates a base mesh that complements our choice of mesh improvement strategy later on. The result is unprecedented robustness and element quality.

Tetrahedral meshing is a hard problem. The strategies found in the literature span a wide range of ideas, from the use of machine learning to predict hard cases [Chen et al. 2012] to the various advancing front methods to generate initial meshes [Alauzet and Marcum 2013; Cuillière et al. 2012; Haimes 2015]. The quality of advancing front outputs can be deceptive: problems are pushed into the interior. Even if the exterior looks perfect, quality in the interior may be arbitrarily poor. We found no reliable advanced front methods suitable for our full-scale comparison.

Surface Envelope. Explicit envelopes have been used to guarantee a bounded approximation error in surface reconstruction. Shen et al. [2004] convert a polygon soup into an implicit representation using a novel interpolation scheme, where a watertight ϵ-isosurface can be extracted for surface approximation purposes. Mandad et al. [2015] create an isotopic surface approximation within a tolerance volume using a modified Delaunay refinement process followed by an envelope-aware and topology-preserving simplification procedure. Our approach uses a similar, implicit, ϵ-envelope to ensure that the tracked surface does not move too far from the input triangle soup.

3 METHOD

We start by defining our problem more precisely. As input we assume a triangle soup, a user-specified tolerance ϵ, and a desired target edge length ℓ. *The goal is to construct an approximately constrained tetrahedralization, that is, a tetrahedral mesh that (1) contains an approximation of the input set of triangles, within user-defined ϵ of the input, (2) has no inverted elements, and (3) edge lengths below user-defined bound ℓ. Mesh quality is optimized while satisfying these constraints.* We call a mesh *valid* if it satisfies the first two properties.

The resulting tetrahedralization can be used for a variety of purposes; most importantly, we can use any definition of the interior of a set of triangles to extract a tetrahedralized volume contained "inside" the input triangle soup.

Throughout this paper, we use the term *surface* to refer to collections of faces, not necessarily manifold, connected, or self-intersection free. Our algorithm tackles this problem in two distinct phases: (1) the generation of a valid mesh, disregarding its geometric quality,

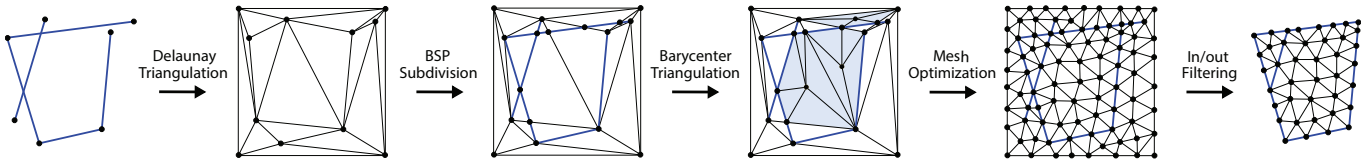


Fig. 2. A diagram illustrating the pipeline of our algorithm in 2D. The points of the original input segments (left) are triangulated using Delaunay triangulation (second left). Each line segment is then split by all triangles that intersect it, constructing a BSP-tree (third left). Each of the resulting convex polygons (colored blue) is divided into triangles by adding a point at its barycenter and connecting it to the vertices of the polygon (third from the right). Local operations are used to improve the quality (second from the right), and finally winding number is used to filter out the elements outside of the domain (right).

representing its coordinates with arbitrary-precision rational numbers and (2) improvement of the geometric quality of its elements and rounding the coordinates of the vertices to floating point numbers, while preserving the validity of the mesh. Decoupling these two sub-problems is the key to the robustness of our algorithm and it is in contrast with the majority of competing methods, which attempt to directly generate a high-quality mesh.

The first phase relies only on operations closed under rational numbers, i.e., the entire computation can be performed exactly if the vertex coordinates are rational, sidestepping all robustness issues (but increasing the computational cost). The second phase uses a hybrid geometric kernel (inspired by [Attene 2017]), allowing us to switch to floating point operations whenever possible to keep the running time sensible (Section 3.4). Our algorithm is thus guaranteed to produce a valid mesh (Phase 1), but we cannot provide any formal bound on its quality (Phase 2): in practice, the quality obtained with our prototype on a dataset of ten thousand *in the wild* models is high (Section 4).

Overview. The algorithm creates a volumetric Binary Space Partitioning (BSP) tree, containing one plane per input triangle and storing its coordinates as exact rational numbers. By construction, the resulting convex (but not necessarily strictly convex) cell decomposition is conforming to the input triangle soup, and a tetrahedral mesh can be trivially created by independently tetrahedralizing each cell (Section 3.1). The volumetric mesh is not only created inside the model, but also around the model, filling a bounding box slightly larger than the input. This allows us to robustly deal with imperfect geometry that contains gaps or self-intersections, postponing the inside/outside segmentation of the space to a later stage in the pipeline. The quality of the mesh is then optimized with a set of local operations to refine, coarsen, swap, or smooth the mesh elements (Section 3.2). These operations are performed only if they do not break a set of invariants that ensure the validity of the mesh at each step. The final mesh is then extracted using winding-number filtering [Jacobson et al. 2013], which is robust to imperfect, real-world input (Section 3.3).

3.1 Generation of a Valid Tetrahedral Mesh

The robust generation of a valid tetrahedral mesh that preserves the faces of an original triangle soup is challenging, even ignoring any quality consideration. Real-world meshes are often plagued by a zoo of defects, including degenerate elements, holes, self-intersection, and topological noise [Attene et al. 2013; Zhou and Jacobson 2016]. Even manually modeled CAD geometry cannot be exported to a clean boundary format, since the most common modeling operations are not closed under spline representation [Farin 2002; Sederberg et al. 2003], unavoidably leading to small "cracks" and self-intersections. Cleaning polygonal meshes or CAD models is a long-standing problem, for which bullet-proof solutions are still elusive [Attene et al. 2013]. We thus propose to use the input geometry as is, and rely on a robust geometrical construction to fill the entire volume with tetrahedra, without committing to the exact topology or geometry of the boundary at this stage, and postponing this challenge to a later stage in the pipeline, after all degeneracies have been removed.

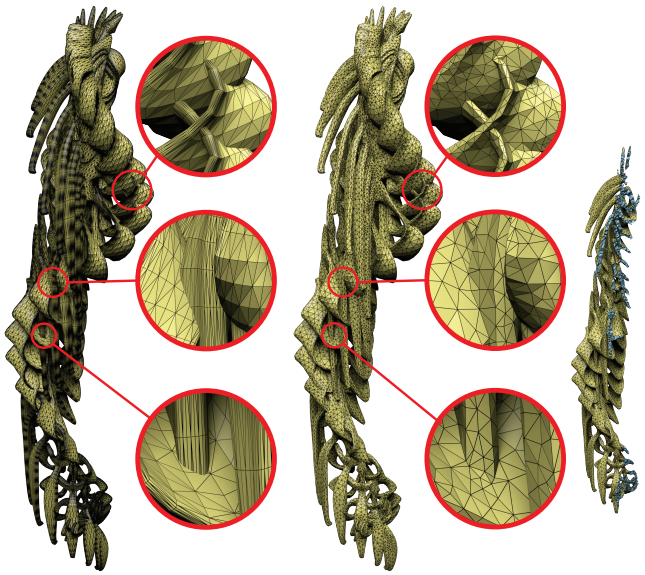


Fig. 3. Self-intersections in the input (left) are automatically handled by our meshing algorithm (right).

BSP-Tree Approach. We build an exact BSP subdivision, using infinite-precision rational coordinates, and only relying on operations closed under this representation. An illustration of the pipeline in 2D is shown in Figure 2: we use a 2D illustration since it is difficult to visualize the effect of operations on tetrahedral meshes in a static figure. In contrast to the surface-conforming Delaunay tetrahedralization [Si 2015], for which designing a robust implementation is challenging (Section 2), the unconstrained version can be robustly implemented with exact rational numbers [Jamin et al. 2015]. We thus create an initial, non-conforming tetrahedral mesh

$\mathcal{M}$, whose vertices are the same as the input triangle soup, using the exact rational kernel in CGAL [Jamin et al. 2015].

The generated tetrahedral mesh does not preserve the input surface, making it unusable for most downstream applications. To enforce conformity, we use an approach inspired by [Joshi and Ourselin 2003], but designed to guarantee a valid output. We consider each triangle of the input triangle soup as a plane, and intersect it with all the tetrahedra in $\mathcal{M}$ that contain it. In other words, we consider each tetrahedron as the root of a BSP cell, and we cut the cell using all the triangles of the input geometry intersecting it. This computation can be performed entirely using rational coordinates, since intersections between planes are closed under rationals, ensuring robustness and correctness even for degenerate input. This polyhedral mesh is converted into a tetrahedral mesh taking advantage of convexity of the cells: we triangulate its faces, add a vertex at the barycenter, and connect it to all the triangular faces on the boundary. Since the only operation necessary is an average of vertex positions, the barycenter can be computed exactly with rationals. As long as at least four input vertices are linearly independent, then *all* convex cells will be non-degenerate, i.e., the resulting tetrahedra connected to the barycenter will also be non-degenerate (though perhaps poor quality). The output mesh is *valid* and exactly conforming to the input triangle soup. Self-intersections in the input are naturally handled by this formulation: they are explicitly meshed, splitting the corresponding triangles accordingly (Figure 3).

3.2 Mesh Improvement

Given a valid tetrahedral mesh represented using rational numbers, we propose an algorithm to improve its quality, and round its vertices to floating point positions, while preserving its validity. We follow the common greedy optimization pipeline based on local mesh improvement operations [Dunyach et al. 2013; Faraj et al. 2016; Freitag and Ollivier-Gooch 1997], but with four important differences:

(1) We explicitly prevent inversions using exact predicates (*Validity Invariant 1*).
(2) We track the surface mesh during the operations, and we only allow operations that keep them within an ϵ distance from the input triangle mesh (inspired by a similar criteria used for surface meshing by [Hu et al. 2017]) (*Validity Invariant 2*).
(3) We directly penalize bad elements in all shapes using a conformal energy which has been recently introduced for mesh parametrization [Rabinovich et al. 2017].
(4) We use a hybrid geometric kernel to reduce the computation time while ensuring correctness and termination, using floating point whenever possible and relying on exact coordinates only where it is strictly necessary.

Invariant 1: Inversions. We disallow every operation introducing inverted tetrahedra whose orientation is negative, using the exact predicates in [Brönnimann et al. 2017] for both rational and floating point coordinates. This ensures an output without inversions, since the algorithm starts from an inversion-free tetrahedral mesh produced by our BSP-tree construction (Section 3.1).

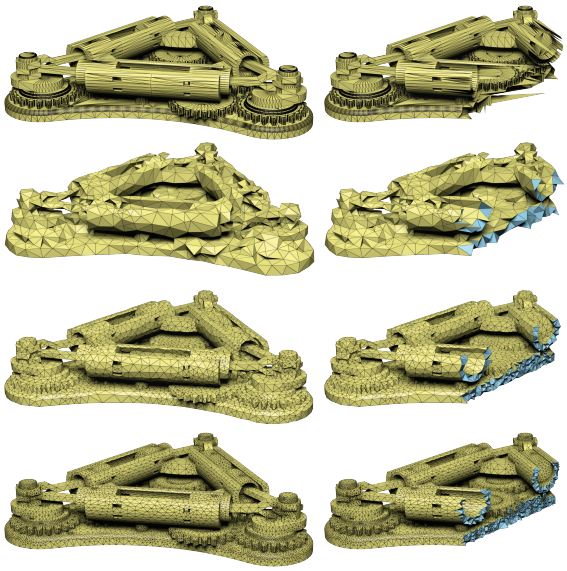


Fig. 4. An oversized ϵ ($\frac{b}{100}$, with b being the bounding box diagonal) creates a tetrahedral mesh (2nd row) that fails to capture the features of the input triangle mesh (1st row). Reducing ϵ to $\frac{b}{300}$ and $\frac{b}{3000}$ increases the geometric fidelity (3rd and 4th row).

Invariant 2: Input Surface Tracking and Envelope. By construction, the tetrahedral mesh produced in Section 3.1 contains an exact representation of all input triangles, in the form of a collection of faces of the tetrahedra. That is, the tetrahedral mesh contains one (or more) tetrahedra whose faces exactly match any given input triangle. We call this collection of faces the *embedded surface*, and all operations performed on the tetrahedral mesh keeps track of it.

To bound the geometric approximation error introduced during the mesh improvement procedure, we only accept operations that keep the faces of the embedded surface at a distance smaller than a user-defined ϵ. Intuitively, this can be depicted as an *envelope* of thickness ϵ built around the input triangle soup. We ensure that the embedded surface is always contained in the envelope at all times by disallowing any operation breaking this invariant (Figure 4).

Quality Measure. As a measure of quality to optimize, we use the 3D conformal energy recently explored in [Rabinovich et al. 2017], which is well-correlated with many common measures of quality (we evaluate the results on a number of measures). It is expressed as:

$$\mathcal{E} = \sum_{t \in T} \frac{\mathrm{tr}(\mathbf{J}_t^T \mathbf{J}_t)}{\det(\mathbf{J}_t)^{\frac{2}{3}}} \tag{1}$$

where $\mathbf{J}_t$ is the Jacobian of the unique 3D deformation that transforms the tetrahedron t into a regular tetrahedron. This energy is oblivious to isotropic scaling, but naturally penalizes needle-like elements, flat and fat elements, slivers, and prevents inversions since it diverges to infinity as an element approaches zero volume. It is also differentiable [Rabinovich et al. 2017], and can be efficiently minimized using Newton or Quasi-Newton iterations [Kovalsky et al. 2016; Rabinovich et al. 2017].

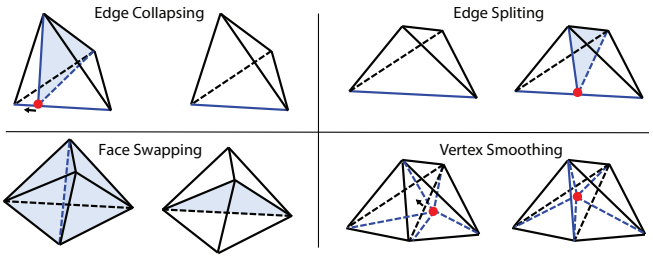


Fig. 5. Overview of the local mesh improvement operations. For face swapping, our algorithm uses 3-2, 4-4, 5-6 bistellar flips [Freitag and Ollivier-Gooch 1997], where 3-2 flip is illustrated here.

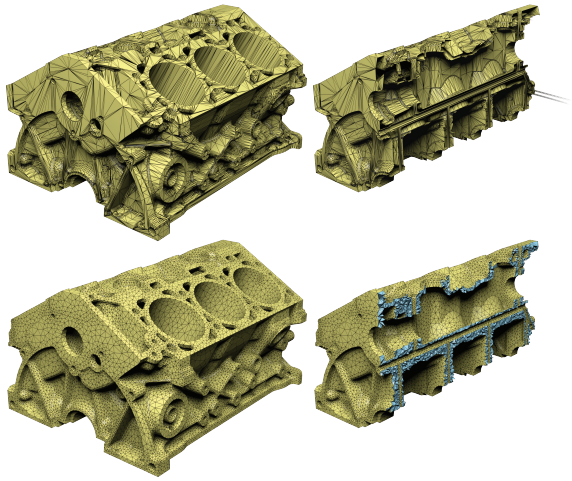


Fig. 6. A low quality triangle mesh exported from a CAD model with OpenCascade (top) is automatically converted into a high-quality tetrahedral mesh by our algorithm (bottom), without requiring any manual cleanup.

Local Operations. We use four local operations for mesh improvement [Faraj et al. 2016; Freitag and Ollivier-Gooch 1997]: edge splitting, edge collapsing, face swapping, and vertex smoothing (Figure 5). These operations only affect a local region of the mesh, and can thus be performed efficiently. We propose an asymmetric optimization scheme: coarsening and optimization operators are applied only if they improve the mesh quality, while the refinement operator is applied until a predefined edge length (user-controlled) is reached, or whenever a region is locked due to the lack of enough degrees of freedom. The rationale behind this strategy is that we want to avoid over-refinement in regions where it is not necessary to improve quality, and we thus add additional vertices only to match the user-provided density or locally if they are necessary to improve the quality. This strategy allows us to produce high-quality meshes even if the input surface has low quality (Figure 6).

We optimize the mesh using 4 passes: (1) splitting (refining), (2) collapsing (coarsening), (3) swapping, and (4) smoothing. We store a target edge length value at the vertices of the tetrahedral mesh, initialized with the user-specified desired edge length ℓ. In (1) each

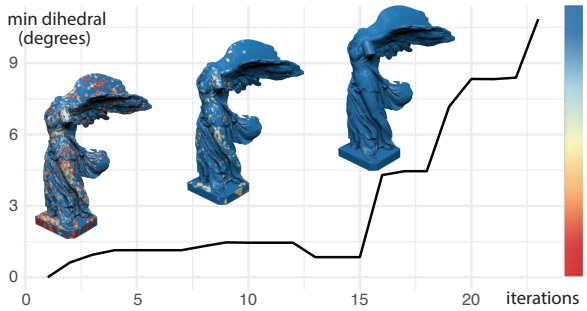


Fig. 7. A mesh generated with the BSP-tree approach is processed by our iterative mesh optimization algorithm. The quality might decrease during the iterations due to the local refinement ignoring quality, but it quickly improves after additional passes of collapsing, swapping, and smoothing.

edge whose length is larger than $\frac{4}{3}$ [Botsch and Kobbelt 2004; Dunyach et al. 2013] times the average of the target edge lengths assigned to its endpoints is split once, and the average is assigned to the new vertex. After (1), the target edge length assigned to a vertex v is divided by 2 if there is a low-quality tetrahedron ($\mathcal{E} > 8$, Equation 1) within its ℓ_v ball, and multiplied by 1.5 otherwise. To ensure that the user-specified density is always reached, we limit the length by the user-specified parameter ℓ. To prevent unnecessary over-refinement in problematic regions, we cap below the length by ϵ. In (2), we collapse an edge if its length is smaller than $\frac{4\ell}{5}$. In (3), we swap faces if they improve the quality. In (4), we smooth all vertices individually minimizing the average of Equation 1 over their one-ring, using Newton's iteration. Only vertices roundable to floats are smoothed, the others are skipped. All these operations are performed only if they do not break any of the invariants described above, and if they increase the mesh quality (with the exception of (1)). In each pass, we use a priority queue to decide the orders of the operations (longest edge first for (1) and (3) and shortest edge first for (2)), except for (4) where the vertices are processed in random order. For (4), we use analytic gradient and Hessian. In Figure 7, we show the effects of the mesh improvement step.

The mesh improvement process stops when either the maximum energy is sufficiently small (default: less than 10) or the maximum number of iteration is reached (default: 80 iterations).

3.3 Interior volume extraction

Note that until this point, our algorithm has not attempted to define a closed surface bounding a volume: the result of the previous stage is a construction of the approximately constrained tetrahedralization, with a possibly nonmanifold, disconnected and open embedded surface.

We use the method proposed in [Jacobson et al. 2013] to address possible imperfections in the embedded surface, by defining an inside-outside function that can be used to extract an interior volume associated with the mesh.

We calculate the winding number of the centroid of each tetrahedron with respect to the embedded surface. If the winding number of the centroid of an element is smaller than 0.5, we consider it outside of the surface and drop it before exporting the mesh. Note that this technique must be applied only after mesh optimization

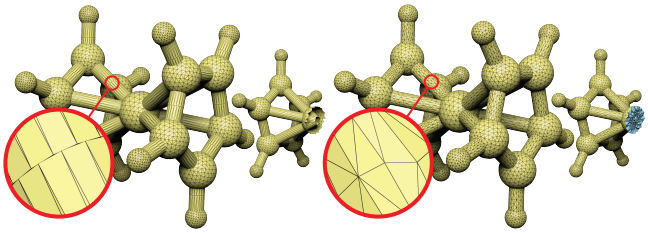


Fig. 8. Any gap or hole in the input geometry (top) is automatically filled by our algorithm (bottom), generating an analysis-ready tetrahedral mesh.

due to numerical reasons: the computation of the winding number cannot be performed in rational numbers and it is numerically unstable close to the surface (where we care the most), due to the use of trigonometric functions.

As a result of this step, both small gaps and large surface holes will be filled according to the induced winding number field (Figures 8 and 11). Consequently, if the input mesh has holes, our algorithms produces a tetrahedral mesh whose surface is not completely inside the ϵ envelope, since the triangles used for hole filling may be outside.

3.4 Technical Detail

Hybrid Kernel. Implementing the mesh optimization with only exact rational numbers to store the position of the vertices is not practical for two reasons: (1) the size of the rational representation grows every time a vertex is modified (dramatically increasing the computation time as the algorithm proceeds, especially in the smoothing step), and (2) rational operations are not supported directly in hardware, and are much slower than floating point operations. We implemented our algorithm using an hybrid geometric kernel, similar in spirit and design to [Attene 2017]. For each vertex, we store its coordinates in exact rational numbers only if any of the incident tetrahedra invert after rounding its vertices to floating point representation. This has two major benefits: it avoids the growth of the rational representations, since it trims their length as soon as it is possible to round a vertex, and reduces the memory consumption. Note that this does not affect the correctness of the algorithm since problematic regions containing almost degenerate elements will continue to use an exact rational representation.

Voxel Stuffing. While guaranteed to produce a valid mesh for any input, the algorithm described in Section 3.1 can (and will) generate poorly-shaped initial cells whose size is different from what the user prescribed, requiring extensive cleanup in the mesh improvement step. To reduce running times, we found it beneficial to preemptively add some proxy points in a regular lattice inside the bounding box of the input triangle soup. To avoid creating degenerate cells, we remove proxy points that are within δ ($\delta > \epsilon$, default: $\delta = \frac{b}{40}$) from the surface. These points are passed to the Delaunay tetrahedralization algorithm (Figure 9), producing a superior starting point that requires fewer local operations to reach a usable quality. In addition to reducing the timing in the optimization stage, this step also localizes the BSP construction around the input surface. We experimentally found that setting the grid edge length to $\frac{b}{20}$ provides the highest benefit, with b being the length of the diagonal of the bounding box.

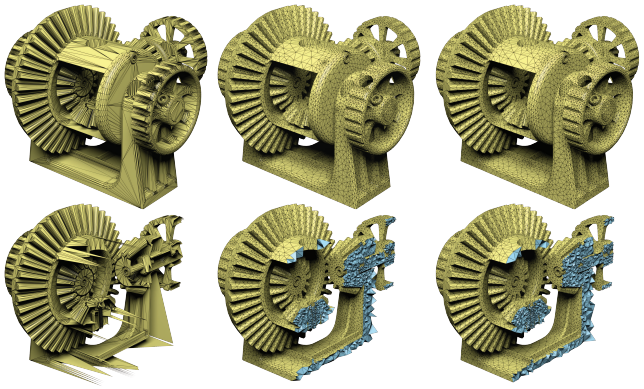


Fig. 9. Voxel stuffing produces a tetrahedral mesh (middle) of quality comparable to a direct BSP-tree construction (right), but reduces the running times from 3292.3 seconds to 2476.6 seconds.

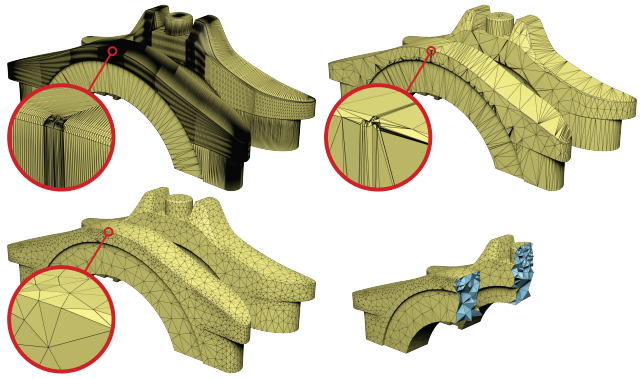


Fig. 10. A heavily tessellated bridge model from Thingi10k (top, left), is simplified by our algorithm, while keeping the surface in the envelope (top, right), and then converted into a tetrahedral mesh (bottom).

Input Simplification. The BSP-tree construction potentially introduces a quadratic number of intersections with respect to the number of faces. This only happens in rare pathological cases and it is not an issue for the majority of real-world models, but we did find two problematic ones over ten thousand in Thingi10k [Zhou and Jacobson 2016] (one of which is shown in Figure 10). In these two models, this issue is sufficiently severe to make the BSP tree mesh larger than 64GB, making our implementation crawl due to memory swapping. We propose a preprocessing step that, while not changing the upper bound complexity of our algorithm, resolves this issue on all meshes we tested it with. The preprocessing tries to: (1) collapse all manifold edges of the input triangle soup, accepting the operations that do not move the surface outside of the envelope and (2) improve the quality of the mesh (in terms of angles) by flipping edges, still keeping the surface in the envelope. This procedure simplifies regions with low curvature, and effectively reduces the number of vertices introduced by the BSP tree. We were not able to construct a synthetic case that breaks this procedure when

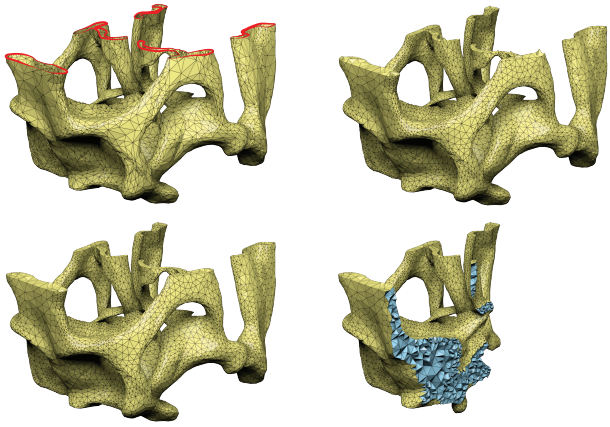


Fig. 11. For an input model with open boundaries (top, left, red lines), we add a reprojection in the smoothing step to preserve them (top, right). To improve the surface quality, we apply Laplacian smoothing to the output faces used to fill the open regions (bottom).

a realistic ϵ is provided. We used this procedure for all our results, since it improves performance also on non-pathological meshes.

Open Boundaries. If the surface contains an open boundary, using only the surface envelope is not always sufficient to ensure a good approximation of the input triangle soup: while unlikely to happen, the boundary is free to move anywhere inside it, potentially moving away from the open boundary, while staying inside the envelope. We address this problem, tracking the open boundaries and reprojecting its vertices back to it in the smoothing step (Figure 11). We consider an edge an open boundary if only one triangle is incident to it.

Envelope Test. Our algorithm heavily relies on testing whether a triangle is contained inside the mesh envelope or not to ensure that the embedded surface stays within the envelope during optimization (Section 3.2). An exact solution would be prohibitively expensive for our purpose [Bartoň et al. 2010; Tang et al. 2009], and we thus use a conservative floating point approximation. Since the approximation error is bounded, our method guarantees that none of the output surface points is outside the envelope.

We implicitly construct the envelope by measuring point-to-mesh distance to the unprocessed input mesh, accelerated by an AABB tree [Lévy 2018; Samet 2005]. To check if an embedded surface triangle face is inside of the envelope, we sample this face using a regular triangular lattice with d as the length of the lattice edge. We also add additional samples on the edges of the face, ensuring a maximal sampling error of $d/\sqrt{3}$ (Figure 12, left). The triangle is considered inside if all the samples are closer than $\hat{\epsilon} = \epsilon - d_{err}$ ($d_{err} = d/\sqrt{3}$), which is a *conservative* envelope. Since the maximal sampling error is bounded by d_{err}, this ensures a correct result, up to floating point rounding. This construction allows us to control the computational cost: a small d means denser sampling and more computational cost but leads to a wider envelope, allowing our algorithm more flexibility in relocating the vertices. Our experiments showed that $d = \epsilon$ ($\hat{\epsilon} = (1-1/\sqrt{3})\epsilon$) is a good compromise.

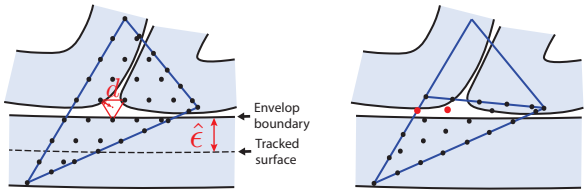
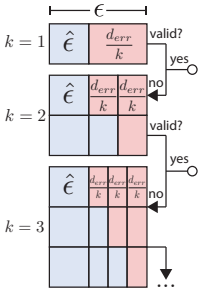


Fig. 12. A triangle face sampled using a triangular lattice has all samples inside the conservative $\hat{\epsilon}$-envelope can have points outside the envelope by at most $d/\sqrt{3}$ (left). Splitting the triangle into two changes the sampling pattern (right), and some samples on one of its sub-faces are now outside the conservative envelope (marked in red).

However, the discrete nature of the sampling introduces a subtle problem: our envelope check is conservative, but not consistent, i.e. reallocating samples on a face of embedded surface by editing its vertices could make it erroneously classified as outside, since some samples might land outside the conservative envelope $\hat{\epsilon}$ (but not outside the user-specified envelope ϵ) (Figure 12, right). This could prevent the optimization algorithm for improving the quality of some regions, since operations might be rejected due to the excessively conservative envelope check. This is a rare occurrence, we observed it on only 3 models over 10k (0.03%).

We propose a robust, yet expensive, solution for these problematic cases: observing that if there are locked elements, enlarging $\hat{\epsilon}$ by d_{err} guarantees that all elements will be free to move again, we increase the sampling density to make enough space for enlarging the envelope, so that locked regions are freed without violating the user-specified envelope ϵ. Let k an integer representing the current stage (the initial stage is denoted by $k = 1$). In stage k: we (1) set the sampling distance to $d_k = d/k$, (2) run the algorithm, and then (3) enlarge the envelope for $k-1$ times by d_{err}/k each time during the geometric optimization (see inset). If a model is still invalid (i.e. the output contains no unroundable vertex) after the geometric optimization in stage k, we then enter into stage $k+1$, rerun the algorithm with a denser sampling, and repeat this procedure until it succeeds.



Across the Thingi10K dataset, 9997 models produced *valid* outputs after stage 1, and the remaining 3 models succeed after stage 2. Since enlarging envelope gives more freedom for moving vertices and cleaning surface, this method can also help to improve quality to some degree: we got 99.98% output tetrahedral meshes have minimal dihedral angle larger than 1 degree with $k = 2$, while this percentage is only 99.52% with $k = 1$.

4 RESULTS

We implemented our algorithm in C++, using Eigen for linear algebra routines, CGAL and GMP for rational computations. The source code of our reference implementation is available at https://github.com/Yixin-Hu/TetWild.

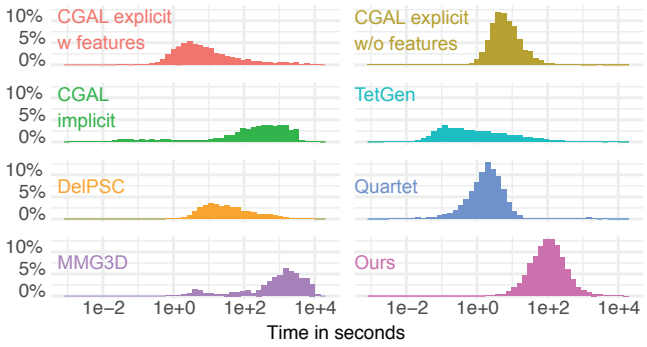


Fig. 13. Comparison of running time.

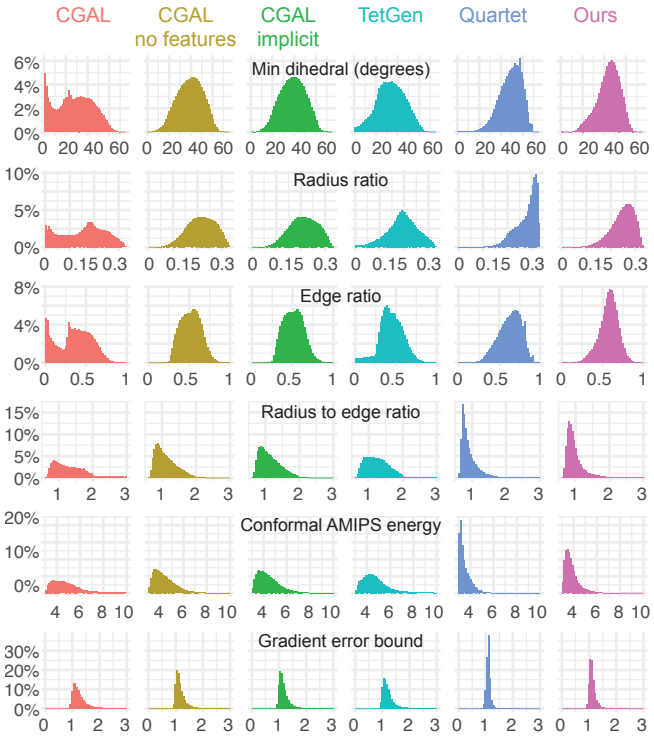


Fig. 14. Comparison of generated mesh quality on Thingi10k dataset. For each software, we show the distribution of 6 common quality measures of all tetrahedra in 1000 randomly sampled meshes generated from Thingi10k dataset. Quality measures: dihedral angle, inscribed/circumscribed sphere radius ratio, conformal AMIPS energy, and normalized Shewchuk's gradient error estimate factor ([Shewchuk 2002b]).

Robustness and Performance. We tested our algorithm and a selection of competing methods over the entire Thingi10k dataset [Zhou and Jacobson 2016]: we show a few examples in Figure 15, report aggregate statistics in Table 2, running times in Figure 13, and output mesh quality in Figure 14. We also report detailed statistics for all models shown in the paper (with the exception of Figure 1) in Table 1. We selected their parameters to make the comparison as fair as possible, and we provide all parameters used in the additional material. **CGAL.** We compared our method with [Jamin et al. 2015] in 3 scenarios: (1) CGAL with polyhedral oracle with feature protection, (2) CGAL with polyhedral oracle without feature protection, and (3) CGAL with implicit surface oracle. (1) and (2) are run using the standard implementation inside CGAL, enabling and disabling feature protection (Section 2), respectively. For (3), we passed an implicit function based on the winding number calculation, used in our filtering. We provide a signed distance field as oracle (computed with the AABB tree in [Jacobson et al. 2016]), and use the winding number [Jacobson et al. 2013] to decide its sign. In all cases, we have observed lower robustness compared to our algorithm. The quality is slightly better for our algorithm. CGAL with the polyhedral oracle is on average 3 to 4 times faster than our algorithm, while CGAL with implicit oracle is much slower: nearly a third of the inputs timed out after 3 hours (Table 2). We show a more detailed comparison of the quality (measured using 6 different criteria) in Figure 16. **Tetgen** [Si 2015] is an order of magnitude faster than our method, but cannot process around half of Thingi10k. It produces meshes with a quality consistently lower than ours, despite introducing more elements. **DelPSC** [Dey and Levine 2008] suffers from robustness problems, successfully processing only around 38%

Table 1. Statistics for the datasets in the paper.

Model Id	Fig.	Input #V	Output #V	Angle	AMIPS	Time(m)
255648	3	91550	61506	4.8/41.3	16.1/4.1	48.8
134705	4	66045	2208	5.6/41.4	11.5/4.1	3.0
134705	4	66045	11341	11.7/46.4	7.8/3.7	10.3
134705	4	66045	470742	10.3/47.3	11.4/3.7	168.5
114029	6	123565	118347	10.3/45.4	9.2/3.7	47.2
376252	7	980051	31734	11.1/45.8	8.0/3.7	10.9
62526	8	7818	25773	8.9/43.7	9.6/3.9	17.7
38416	9	120172	87648	10.2/46.3	8.0/3.7	44.6
996816	10	76111	12663	0.02/45.0	1625.4/4.0	747.7
48354	11	10945	21211	10.5/45.8	8.0/3.7	3.4
486859	15	14629	15011	10.0/45.3	9.3/3.7	5.3
42155	15	24646	7248	13.3/45.4	7.3/3.7	2.1
78481	15	298370	11385	12.7/46.4	7.9/3.7	3.9
551021	15	174066	51011	10.2/46.1	9.4/3.7	16.5
488049	15	23036	3574	13.0/43.2	7.8/4.0	1.3
47076	15	768	5491	9.7/44.7	9.6/3.8	1.0
964933	16	148	4991	10.0/44.5	8.3/3.8	1.2
1036403	17	87046	46220	10.5/45.1	8.1/3.8	20.3
1036403	17	87046	202846	12.4/50.1	7.7/3.5	162.7
252683	18	906835	34721	10.0/44.5	8.2/3.8	14.1
252683	18	906835	119087	10.1/46.4	8.0/3.7	113.4
78211	19	320	2042	11.3/34.2	9.9/4.6	0.5
78211	19	320	8661	9.3/43.5	10.1/3.9	14.2
63465	20	592	6238	14.1/44.9	8.2/3.8	0.9
76538	21	14169	10098	12.0/44.9	7.9/3.8	3.9
1065032	22	48506	27362	8.5/45.4	9.4/3.8	9.2
1036658	23	4244	3713	12.3/43.7	7.9/3.8	1.4
Bunny	24	11247	38326	7.7/43.8	9.3/3.9	7.2
Bunny	24	11247	87359	9.9/43.0	8.1/4.0	20.8
1505037	25	19218	37782	10.2/44.2	8.0/3.9	16.8

Note: From left to right: Thingi10k model ID, figure where it appears, number of input vertices, number of output vertices, dihedral angle (min/avg), AMIPS energy (Equation 1) (max/avg), running time in minutes.

Fig. 15. Comparison with state-of-art tetrahedralization algorithms. The number close to each model is the minimal dihedral angle.

Table 2. Comparison of code robustness and performance on the Thingi10k dataset

Software	Success rate	Out of memory(>32GB)	Time exceeded(>3h)	Algorithm limitation	Average time(s)
CGAL (explicit, w features)	57.2%	5.4%	15.7%	21.7%	160.2
CGAL (explicit, wo features)	79.0%	0.0%	0.0%	21.0%	11.7
CGAL (implicit, wo features)	55.7%	0.0%	32.6%	11.7%	997.3
TetGen	49.5%	0.1%	1.7%	48.7%	32.3
DelPSC	37.1%	0.0%	31.1%	31.7%	174.8
Quartet	87.2%	0.0%	0.0%	12.8%	15.3
MMG3D	56.2%	1.2%	10.8%	31.8%	2182.3
Ours	99.9*%	0.0%	0.1%	0.0%	360.0

Note: The maximum resource allowed for each model are 3 hours and 32GB of memory. *Our method exceeds the 3h time on 11 models. If 27 hours of maximal running time are allowed, our algorithm achieves 100% success rate.

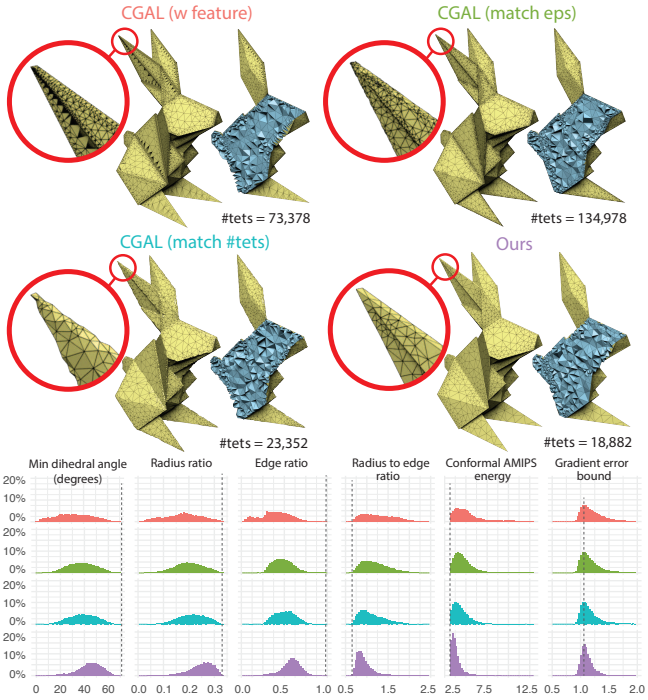


Fig. 16. (Top): With the same meshing parameters ($\epsilon = b/2000$ and $\ell = b/20$), CGAL's algorithm with and without feature protection (top row) used more than 4 and 7 times the number of tets than ours (second row right) respectively. When using roughly the same number of tets, CGAL's result (second row left) struggles to preserve sharp features. (Bottom): Histograms of various tet quality measures for all tets generated from CGAL and our algorithm. The dotted lines indicate the ideal quality values computed on a regular tetrahedron. Note that our results (bottom row) have better quality in all measures.

of Thingi10k. The quality is consistently lower than ours. **Quartet** [Bridson and Doran 2014] is the most robust competing method, with a success rate of 88%. It unfortunately struggles to preserve thin features, and often uses a much higher element count than our method.

Parameters. Our algorithm requires two parameters: the target edge length ℓ, which controls the density of the output mesh, and

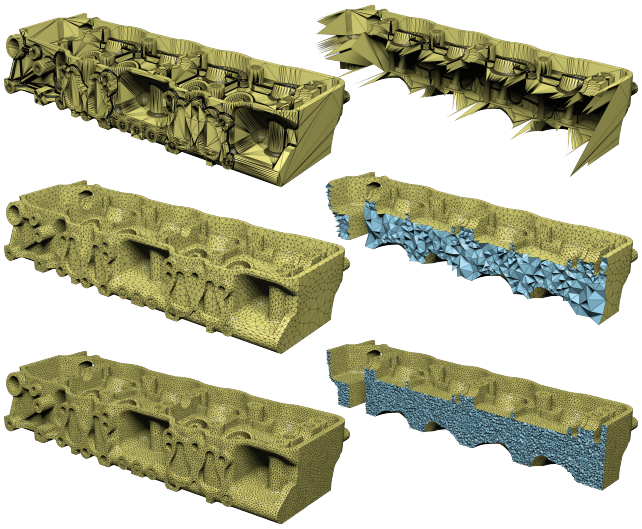


Fig. 17. ℓ controls the density of the output mesh. Input (top), $\ell = b/20$ (middle) and $\ell = b/150$ (bottom).

the maximal Hausdorff distance bound ϵ, which controls the geometric faithfulness of the result. For all our experiments (except where noted otherwise) we used $\ell = b/20$ and $\epsilon = b/1000$, where b is the length of the diagonal of the bounding box of the input. The parameter ℓ controls the mesh density directly (Figure 17), while ϵ does it indirectly. Prescribing a small ϵ forces the algorithm to refine more to enforce the tighter bound. Providing a larger ϵ allows our algorithm to get close to the user-prescribed lenghts (Figure 18).

Spatially Varying Sizing Field. By replacing the uniform target edge length ℓ with a spatially varying function $\ell(p)$, our algorithm can be extended to create graded meshes. Figure 19 illustrates a result with target edge length smoothly varying from coarse to fine in a single model. Note that the output mesh quality remains high despite the large change in the sizing field.

Surface Repair. Our algorithm can be used as an effective mesh repair tool for closed surfaces by creating a tetrahedral mesh of their interior, and then extracting its boundary. Self-intersections are robustly resolved when constructing the BSP-tree, degeneracies

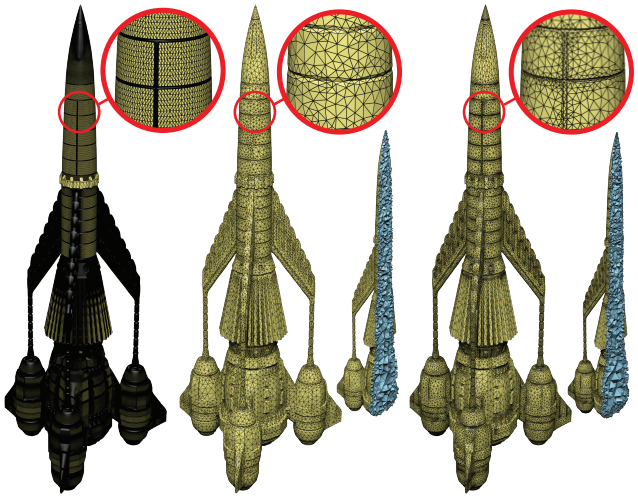


Fig. 18. ϵ bounds the maximal distance between the input and output mesh. Input (left), $\epsilon = b/1000$ (middle) and $\epsilon = b/3000$ (right).

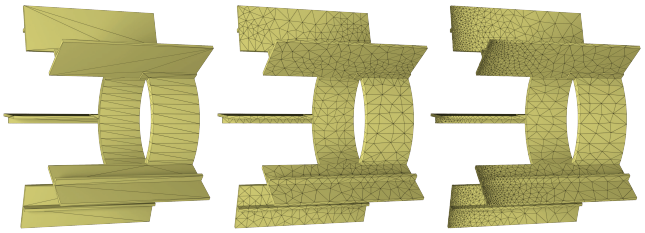


Fig. 19. Example for spatially varying sizing field using background mesh. Input (left), output tetrahedral mesh without sizing control (middle), and output tetrahedral mesh with sizing field applied (right).

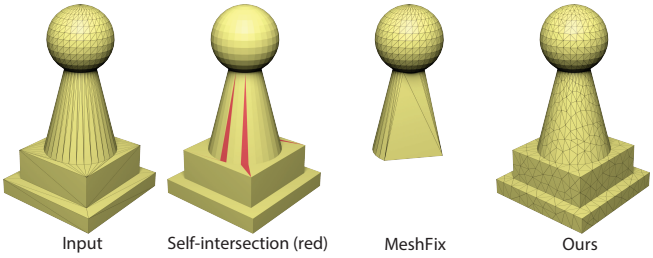


Fig. 20. A self-intersecting triangle soup, is cleaned using meshfix by removing the base. Our algorithm instead creates a tetrahedral mesh of its interior, whose boundary corresponds to a clean triangle mesh of the pawn.

are removed by the mesh improvement step, surface gaps/holes are filled based on generalized winding number, and the output surface is trivially the boundary of a valid volume. While computationally more expensive than alternative methods that only work on the surface, our technique can robustly handle extremely challenging cases. In Figure 20, we compare our method to MeshFix [Attene 2010] on a self-intersecting chess pawn.

Finite Element Method Validation. We demonstrate that our algorithm can be used as a black box to solve PDEs on the entire

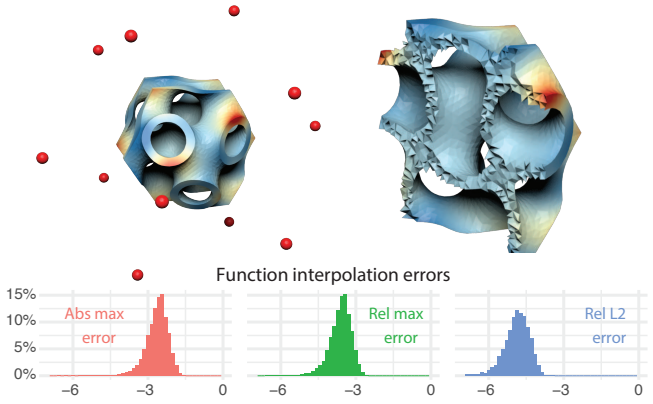


Fig. 21. We test our generated tet meshes by solving a harmonic PDE using finite element method with linear elements. For each model in Thingi10K, we compare the computed solution with the ground truth (radial basis functions with kernel $1/r$ centered at the red spheres). We show the absolute max error, relative max error, and relative L_2 error histograms (log scale) in the bottom row.

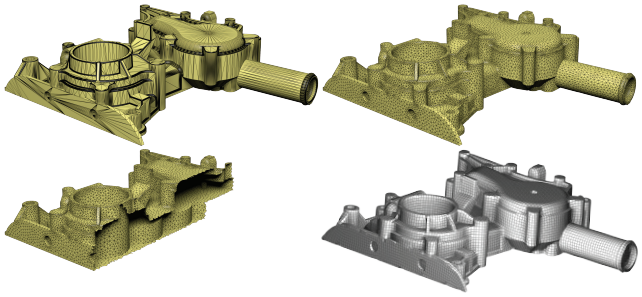


Fig. 22. Our algorithm can be used to bootstrap quadrilateral remeshing.

Thingi10k dataset. We normalize all our output meshes to fit in the unit cube and create an analytic volumetric harmonic function by summing 12 radial kernels $(1/r)$, placed randomly on a sphere centered at the origin of radius $1.5b$. This function is sampled on the boundary of the mesh and used as a boundary condition for a Poisson problem, solved using [Jacobson et al. 2016]. We successfully solve this PDE over all models, and we report a sample solution and the histograms of L^2 and L^∞ errors with respect to the analytic solution evaluated on the internal nodes in Figure 21.

Structured Meshing. Structured meshing algorithms [Bommes et al. 2012] usually rely on an existing clean boundary representation of the geometry (triangle meshes in 2D and tetrahedral meshes in 3D) to generate a structured mesh. Our algorithm can be used to convert triangle soups into meshes suitable for remeshing. We show the examples of quadrilateral meshing using [Jakob et al. 2015] in Figure 22 and hexahedral-dominant meshing [Gao et al. 2017] in Figure 23.

Noise Stress-Test. We stress test our method under geometrical noise (Figure 24), by randomly displacing its vertices using Gaussian noise. Even in this extreme case our algorithm produces meshes close to the noisy input and have a large minimal dihedral angle.

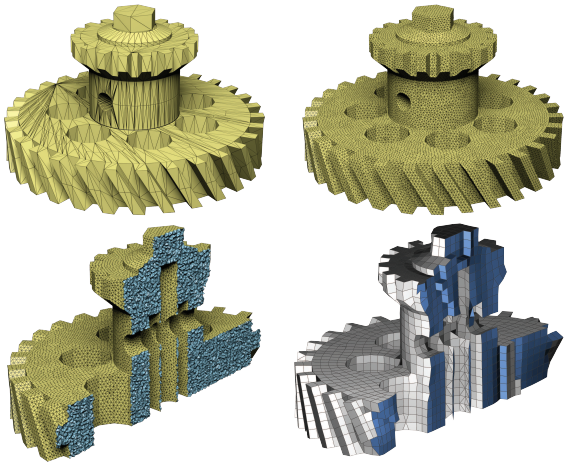


Fig. 23. Our algorithm can be used to bootstrap hex-dominant remeshing.

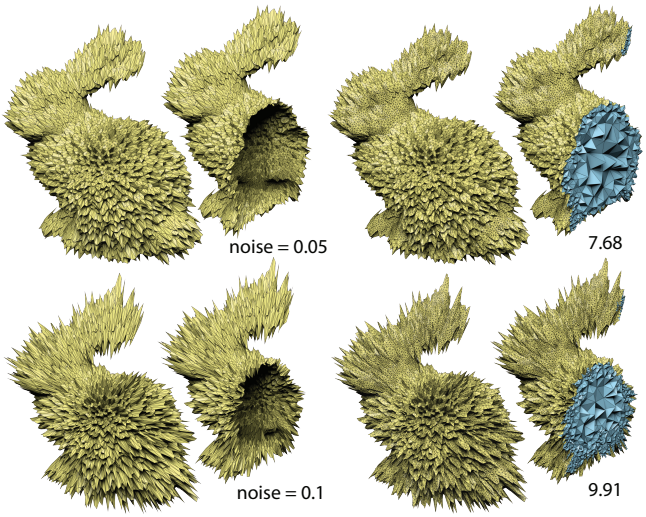


Fig. 24. Our algorithm is robust to geometrical noise. The numbers denote the minimal dihedral angle of output meshes.

Meshing for Multimaterial Solids. Our algorithm naturally supports the generation of tetrahedral meshes starting from multiple enclosed surfaces by simply skipping the filtering step (Section 3.3), as shown in Figure 25.

5 LIMITATIONS AND CONCLUDING REMARKS

Our algorithm handles sharp features in a soft way: they are present in the output, but their vertices could be displaced, causing a straight line to zigzag within the envelope. While this is acceptable for most graphics applications, extending our algorithm to support exact preservation of sharp features is an interesting research direction that we plan to pursue. We demonstrated that our algorithm can be used as a mesh repair tool, but it is, however, limited to closed surfaces: extending it to support mesh repair over shells is an interesting and challenging problem. Our single threaded implementation

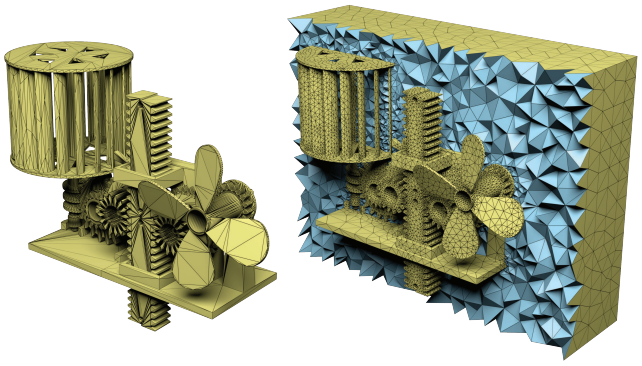


Fig. 25. The volume around a complex mechanical piece is automatically meshed by our algorithm, preserving the surface of the embedded object.

is slower than most competing methods: since most steps of our algorithm are local, we believe that a performance boost could be achieved by developing a parallel (and possibly distributed) version of our approach.

To conclude, we presented an algorithm to compute *approximately constrained tetrahedralizations* from triangle soups. Our algorithm can robustly process thousands of models without parameter tuning or manual interaction, opening the door to black-box processing of geometric data.

ACKNOWLEDGEMENTS

We are grateful to Jérémie Dumas for illuminating discussions, to Wenzel Jakob for the Mitsuba renderer, and to Thingiverse and the Stanford 3D Scanning Repository for the datasets. This work was supported in part by NSF CAREER award 1652515, NSF grant (IIS-1320635 & DMS-1436591), NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938), Canada Research Chair award, Connaught Fund, and a gift from Adobe Systems and nTopology Inc..

REFERENCES

Frédéric Alauzet and David L. Marcum. 2013. A Closed Advancing-Layer Method with Changing Topology Mesh Movement for Viscous Mesh Generation. In *22nd International Meshing Roundtable, IMR 2013, October 13-16, 2013, Orlando, FL, USA.*

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

Marco Attene. 2010. A Lightweight Approach to Repairing Digitized Polygon Meshes. *Vis. Comput.* 26, 11 (Nov. 2010), 1393–1406.

Marco Attene. 2017. ImatiSTL - Fast and Reliable Mesh Processing with a Hybrid Kernel. In *LNCS on Transactions on Computational Science XXIX - Volume 10220.* Springer-Verlag New York, Inc., New York, NY, USA, 86–96.

Marco Attene, Marcel Campen, and Leif Kobbelt. 2013. Polygon Mesh Repairing: An Application Perspective. *ACM Comput. Surv.* 45, 2, Article 15 (March 2013), 33 pages.

Michael Bartoň, Iddo Hanniel, Gershon Elber, and Myung-Soo Kim. 2010. Precise Hausdorff distance computation between polygonal meshes. *Computer Aided Geometric Design* 27, 8 (2010), 580 – 591. Advances in Applied Geometry.

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

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2012. State of the Art in Quad Meshing. In *Eurographics STARS.*

Mario Botsch and Leif Kobbelt. 2004. A Remeshing Approach to Multiresolution Modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04).* ACM, New York, NY, USA, 185–192.

Robert Bridson and Crawford Doran. 2014. Quartet: A tetrahedral mesh generator that does isosurface stuffing with an acute tetrahedral tile. https://github.com/crawforddoran/quartet.

Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2017. 2D and 3D Linear Geometry Kernel. In *CGAL User and Reference Manual* (4.11 ed.). CGAL Editorial Board. http://doc.cgal.org/4.11/Manual/packages.html#PkgKernel23Summary

Jonathan R. Bronson, Joshua A. Levine, and Ross T. Whitaker. 2012. Lattice Cleaving: Conforming Tetrahedral Meshes of Multimaterial Domains with Bounded Quality. In *Proceedings of the 21st International Meshing Roundtable, IMR 2012, October 7-10, 2012, San Jose, CA, USA*. 191–209.

Graham F Carey. 1997. *Computational grids: generations, adaptation & solution strategies*. CRC Press.

Long Chen and Jin-chao Xu. 2004. Optimal delaunay triangulations. *Journal of Computational Mathematics* (2004), 299–308.

Xiaoshen Chen, Dewen Peng, and Shuming Gao. 2012. SVM-Based Topological Optimization of Tetrahedral Meshes. In *Proceedings of the 21st International Meshing Roundtable, IMR 2012, October 7-10, 2012, San Jose, CA, USA*. 211–224.

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.

Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. 2012. *Delaunay mesh generation*. CRC Press.

L Paul Chew. 1989. Constrained delaunay triangulations. *Algorithmica* 4 (1989).

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.

Jean-Christophe Cuillière, Vincent François, and Jean-Marc Drouet. 2012. Automatic 3D Mesh Generation of Multiple Domains for Topology Optimization Methods. In *Proceedings of the 21st International Meshing Roundtable, IMR 2012, October 7-10, 2012, San Jose, CA, USA*. 243–259.

Tamal K Dey and Joshua A Levine. 2008. DelPSC: a Delaunay mesher for piecewise smooth complexes. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*. ACM, 220–221.

Crawford Doran, Athena Chang, and Robert Bridson. 2013. Isosurface Stuffing Improved: Acute Lattices and Feature Matching. In *ACM SIGGRAPH 2013 Talks (SIGGRAPH '13)*. ACM, New York, NY, USA, Article 38, 1 pages.

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 - Short Papers*, M.-A. Otaduy and O. Sorkine (Eds.). The Eurographics Association.

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

David Eppstein. 2001. Global optimization of mesh quality. *Tutorial at the 10th Int. Meshing Roundtable* (2001).

Noura Faraj, Jean-Marc Thiery, and Tamy Boubekeur. 2016. Multi-material Adaptive Volume Remesher. *Comput. Graph.* 58, C (Aug. 2016), 150–160.

Gerald Farin. 2002. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. In *Curves and Surfaces for CAGD* (fifth edition ed.), Gerald Farin (Ed.). Morgan Kaufmann, San Francisco.

Lori A. Freitag and Carl Ollivier-Gooch. 1997. Tetrahedral mesh improvement using swapping and smoothing. *Internat. J. Numer. Methods Engrg.* 40, 21 (1997).

Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017. Robust Hex-dominant Mesh Generation Using Field-guided Polyhedral Agglomeration. *ACM Trans. Graph.* 36, 4, Article 114 (July 2017), 13 pages.

Abel Gargallo-Peiró, Xevi Roca, Jaime Peraire, and Josep Sarrate. 2013. Defining Quality Measures for Validation and Generation of High-Order Tetrahedral Meshes. In *Proceedings of the 22nd International Meshing Roundtable, IMR 2013, October 13-16, 2013, Orlando, FL, USA*. 109–126.

Robert Haimes. 2015. MOSS: multiple orthogonal strand system. *Eng. Comput. (Lond.)* 31, 3 (2015), 453–463.

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

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust Inside-outside Segmentation Using Generalized Winding Numbers. *ACM Trans. Graph.* 32, 4, Article 33 (July 2013), 12 pages.

Alec Jacobson, Daniele Panozzo, et al. 2016. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)* 34, 6 (Nov. 2015).

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.

Bhautik J. Joshi and Sébastien Ourselin. 2003. BSP-Assisted Constrained Tetrahedralization. In *Proceedings of the 12th International Meshing Roundtable, IMR 2003, Santa Fe, New Mexico, USA, September 14-17, 2003*. 251–260.

Bryan Matthew Klingner and Jonathan Richard Shewchuk. 2007. Agressive Tetrahedral Mesh Improvement. In *Proceedings of the 16th International Meshing Roundtable*. Seattle, Washington, 3–23. http://graphics.cs.berkeley.edu/papers/Klingner-ATM-2007-10/

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Trans. Graph.* 35, 4, Article 134 (July 2016), 11 pages.

François Labelle and Jonathan Richard Shewchuk. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 57.

Bruno Lévy. 2018. Geogram. http://alice.loria.fr/index.php/software/4-library/75-geogram.html.

Manish Mandad, David Cohen-Steiner, and Pierre Alliez. 2015. Isotopic Approximation Within a Tolerance Volume. *ACM Trans. Graph.* 34, 4, Article 64 (July 2015), 12 pages. https://doi.org/10.1145/2766950

Marek Krzysztof Misztal and Jakob Andreas Bærentzen. 2012. Topology-adaptive interface tracking using the deformable simplicial complex. *ACM Trans. Graph.* 31, 3 (2012), 24:1–24:12.

Neil Molino, Robert Bridson, and Ronald Fedkiw. 2003. Tetrahedral mesh generation for deformable bodies. In *Proc. Symposium on Computer Animation*.

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.

Steven J Owen. 1998. A survey of unstructured mesh generation technology.. In *IMR*.

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.

Jean-François Remacle. 2017. A two-level multithreaded Delaunay kernel. *Computer-Aided Design* 85 (2017), 2–9.

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

Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Erich Schönhardt. 1928. Über die zerlegung von dreieckspolyedern in tetraeder. *Math. Ann.* 98, 1 (1928), 309–312.

Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. 2003. T-splines and T-NURCCs. *ACM Trans. Graph.* 22, 3 (July 2003), 477–484.

Donald R Sheehy. 2012. New Bounds on the Size of Optimal Meshes. *Comput. Graph. Forum* 31, 5 (2012).

Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. 2004. Interpolating and Approximating Implicit Surfaces from Polygon Soup. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press, 896–904. http://graphics.cs.berkeley.edu/papers/Shen-IAI-2004-08/

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

Jonathan Richard Shewchuk. 2002b. *What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures*. Technical Report. In Proc. of the 11th International Meshing Roundtable.

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.

Min Tang, Minkyoung Lee, and Young J. Kim. 2009. Interactive Hausdorff Distance Computation for General Polygonal Models. *ACM Trans. Graph.* 28, 3, Article 74 (July 2009), 9 pages.

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.

Qingnan Zhou and Alec Jacobson. 2016. *Thingi10K: A Dataset of 10,000 3D-Printing Models, https://ten-thousand-models.appspot.com*. Technical Report. New York University.