Surface chamfering for robust tetrahedral meshing

LORENZO DIAZZI, IMATI - CNR, Italy
JIACHENG DAI, New York University, USA
DANIELE PANOZZO, New York University, USA
MARCO ATTENE, IMATI - CNR, Italy

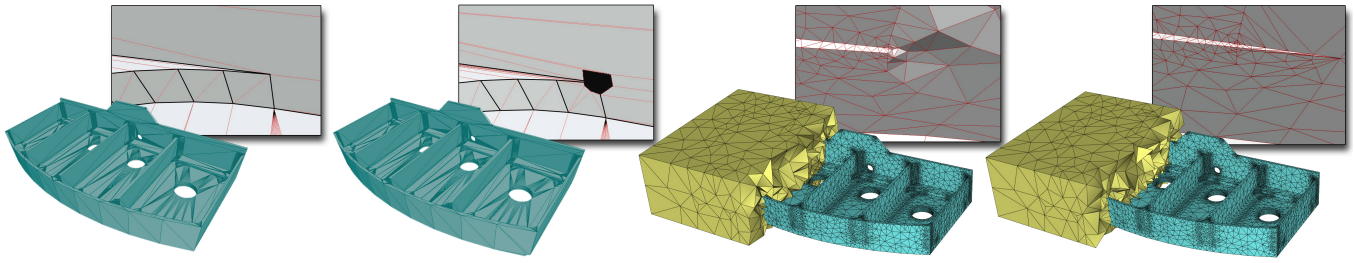


Fig. 1. An input polyhedral surface made of polygonal faces having acute angles, and where each face has an arbitrarily low-quality triangulation (left). Our chamfering algorithm removed all the acute angles from the faces of the polyhedron (mid-left). The volume of this modified surface is meshed with tetrahedra with all face angles greater than 14 degrees (mid-right). Finally, our algorithm provides exact conformance by allowing a few tetrahedra with smaller angles (right).

We present an algorithm that produces high quality tetrahedral meshes conforming with input polyhedra. Our meshing algorithm is based on Ruppert's Delaunay refinement where convergence is guaranteed thanks to a novel *chamfering* approach that removes all acute angles from the input. On such a modified input Delaunay refinement produces a Delaunay tetrahedrization where all the faces have bounded angles. The input portions that were removed by the chamfering are re-inserted in this tetrahedrization to achieve exact conformance at the cost of a small number of bad-shaped tetrahedra near the formerly acute input angles. Numerical robustness is guaranteed along all the phases thanks to a clever use of modern indirect geometric predicates and the definition of a new type of implicit point to represent Steiner vertices on the input faces. In practice, our prototype implementation produces meshes having a quality comparable to the state-of-the-art `tetgen` software: while `tetgen` fails on 37% of the 3942 valid models in the Thingi10k dataset, our method succeeds on **all** of them.

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

ACM Reference Format:
Lorenzo Diazzi, Jiacheng Dai, Daniele Panozzo, and Marco Attene. 2026. Surface chamfering for robust tetrahedral meshing. *ACM Trans. Graph.* 45, 4, Article 148 (July 2026), 15 pages. https://doi.org/https://doi.org/10.1145/3811395

Authors' addresses: Lorenzo Diazzi, IMATI - CNR, Italy, lorenzo.diazzi@ge.imati.cnr.it; Jiacheng Dai, New York University, USA, jd4705@nyu.edu; Daniele Panozzo, New York University, USA, panozzo@nyu.edu; Marco Attene, IMATI - CNR, Italy, marco.attene@ge.imati.cnr.it.

ACM 0730-0301/2026/7-ART148
https://doi.org/https://doi.org/10.1145/3811395

1 INTRODUCTION

Simplicial meshing algorithms (triangulation of polygons and tetrahedralization of polyhedra) are widely used in graphics, scientific computing, and geometry processing to discretize the interior of geometric shapes, as this is a required step for many downstream applications, including solutions of differential equations, modeling, and rendering.

This topic has been heavily studied since the 1980s, with two main families of methods: (1) boundary-conforming algorithms, which tesselate the interior while preserving the boundary geometry exactly, possibly splitting the elements of the boundary, and (2) boundary-approximating algorithms, which approximate the input boundary within a prescribed tolerance.

While the former is usually preferred due to their ease of use in transferring boundary conditions from the input surface to the output volume and due to their higher accuracy in representing the boundary, provably robust implementations have been elusive until very recently. [Diazzi et al. 2023] proposes the first provably robust implementation of the Constrained Delaunay Tetrahedrization (CDT) algorithm, which, however, has the disadvantage of creating low-quality elements. Existing methods of this family that provide reasonable quality [Si 2015] are unfortunately unreliable and often fail due to their use of floating point arithmetic (Section 8). The much simpler problem of computing boundary-approximating meshes has been robustly tackled by either using an implicit intermediate representation [Alliez et al. 2009] or an envelope [Hu et al. 2018]: in both cases, a geometric error is introduced over the entire boundary and the resulting meshes are more challenging to use in downstream applications.

Chamfering. We propose a hybrid approach that strikes an exciting balance between these two, until now disjoint, strategies: we locally modify the input surface in a highly controlled way by *cutting*

all acute angles from the input, thus introducing a localized error. We then propose a provably robust Delaunay refinement algorithm which is guaranteed to produce a graded mesh of tetrahedra with bounded absolute face angles. Most importantly, the angle bound does not depend on the angle of the input triangle mesh. To the best of our knowledge, no previous algorithm provides such guarantees.

Robustness is guaranteed along the whole process thanks to the use of novel implicit points to represent both the cuts and the Steiner points, so that any new point splitting an edge is exactly on the edge and any new point splitting a face is exactly on the face, with no rounding error. Indirect predicates [Attene 2020b] are used to efficiently and robustly analyze these points.

Exact Boundary Preservation. The downside of this approach is that we preserve the input surface everywhere except for a small region around each acute vertex or edge. For applications that do not require exact conformance, users can set a maximum *chamfering radius* to bound the error introduced. For applications where exact conformance is necessary, we compute a CDT of the original input surface endowed with the additional Steiner points computed by our algorithm to obtain a graded mesh with good average quality, but losing the angle bound. We show that this algorithm enjoys the following properties: theoretical guarantees, exact preservation of the input geometry and, experimentally, high average quality. We believe this approach has a high potential for many practical applications, especially when paired with solver strategies to decouple mesh quality from solution quality [Schneider et al. 2018].

Contributions. Our contributions are: (1) A novel chamfering strategy and Delaunay refinement algorithm that creates meshes with bounded quality independently from the quality of the input. (2) A hybrid approach combining the Steiner points computed by the Delaunay refinement with the input surface that trades off the theoretical angle bound with a guaranteed 100% conforming tetrahedrization. (3) An effective implementation using indirect predicates, leading to runtimes and resulting meshes comparable to existing non-robust Delaunay refinement algorithms. (4) A 2D version of both algorithms for the much simpler, yet practically useful, 2D case of filling a polygon with triangles.

Our prototype implementation succeeds on all the 3942 valid models of the Thingi10k dataset [Zhou and Jacobson 2016] and the full source code is available at https://github.com/MarcoAttene/DelOptim. As for any Delaunay refinement method, quality guarantees regard face angles only, which have provably bounded values, but not dihedral angles which can be arbitrarily small.

2 BACKGROUND AND RELATED WORK

In the remainder, the term *meshing* indicates the process of subdividing a compact subset of the plane (or space) into a collection of simple elements. Such a subset is the *domain* of the mesh. This paper deals with domains having a piecewise-linear boundary in two or three dimensions, that is, polygons in 2D or polyhedra in 3D. In particular, we deal with *simplicial* meshing where elements are triangles or tetrahedra.

Within such a context, existing algorithms can be classified based on how they deal with the domain boundary, and can be either approximated or boundary *conforming*.

2.1 Approximated Methods

We refer to [Hu et al. 2018] for a complete overview of this family of methods, and we provide here only a brief overview of the main approaches.

Grid-Based. Instead of constructing a mesh from scratch, many methods propose to start from a background mesh (often a regular grid) and insert the input triangles (or an implicit distance field) into the background mesh [Bridson and Doran 2014; Bronson et al. 2013; Doran et al. 2013; Labelle and Shewchuk 2007; Molino et al. 2003]. These methods can be implemented robustly, but either lose details depending on the background mesh resolution (if the cells outside the object are discarded) or create potential low quality close to the input boundary (if the cells are cut with the input surface).

Envelope. High-quality elements can be obtained by relaxing the boundary approximation, requiring the output mesh to be within a given distance to the input. This idea has been proposed in [Shen et al. 2004] for implicit surfaces and in [Mandad et al. 2015] to create a surface approximation within a tolerance volume using a modified Delaunay refinement process. TetWild [Hu et al. 2018] and its followups [Hu et al. 2019, 2020] first insert the input triangle mesh into a background mesh exactly using rational computation and then optimize it with local operations while preventing from moving outside a given envelope.

Differently from these approximating methods, our approach preserves the input surface exactly everywhere. Alternatively, if exact surface preservation is not required, our method can provide a provable quality guarantee at the cost of a controllable distortion.

2.2 Boundary Conforming Methods

A mesh is boundary conformal if its boundary exactly coincides with the boundary of the input domain. Conformity may be only geometrical, meaning that the geometric realization of the two boundaries coincide while their simplicial subdivisions can be different.

Constrained Delaunay Triangulation. In 2D, constrained Delaunay triangulations (CDTs) [Chew 1987] can be used to produce conforming meshes. The same concept can be extended to 3D [Diazzi et al. 2023; Si 2015] though not all domains admit a CDT unless the input is enriched with additional Steiner points. In 2D one can even create conforming Delaunay triangulations [Shewchuk 2000] by adding Steiner points so that the empty circumcircle property holds for all the triangles. Extensions to higher dimensions [Hudson et al. 2006] and more general triangulations [Shewchuk 2008] also exist.

A limitation of CDT, is that the quality of the elements is uncontrolled, often leading to wide ranges of angles and element sizes. While this is acceptable in some applications, others require well-shaped elements, for example in simulation [Wördenweber 1984]. Broadly speaking, a triangle is well-shaped if it has no extreme angles (i.e. nearly zero or nearly flat). A (possibly constrained) Delaunay triangulation can be enriched with additional Steiner points

to improve the shape of its triangles while maintaining conformity. This process is known as Delaunay refinement.

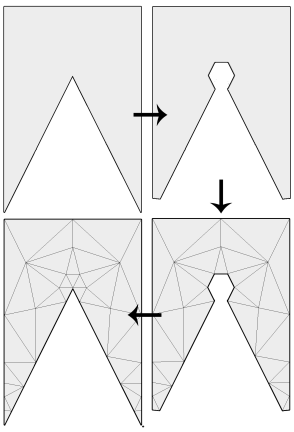
Delaunay Refinement. The basic idea of Delaunay refinement is removing one *bad* triangle at a time by inserting its circumcenter and updating the triangulation to account for it. Ruppert's method [Ruppert 1995] starts with a Delaunay triangulation of the vertices, and iteratively splits all the *encroached* boundary edges at their midpoints (an edge is encroached if its diametral circle encloses a vertex different from its two endpoints). This creates a conforming Delaunay triangulation with possible bad triangles. Each bad triangle is removed by inserting its circumcenter, unless one of the boundary edges is encroached by this circumcenter. In that case, the circumcenter is not inserted and, instead, the encroached edge is split. Chew's method [Chew 1993] is slightly different as it refines a constrained Delaunay triangulation of the input. Upon insertion of the circumcenter c of a triangle t, the algorithm checks whether c is on the opposite side of a boundary edge when viewed from t. If that is the case, c is not inserted, the midpoint of the obstructing edge is inserted instead, and any previously inserted circumcenter in the diametral circle of that edge is removed.

In both cases, a triangle is considered to be *bad* if it has an angle smaller than a prescribed threshold δ. If the domain boundary has no acute angles, Ruppert's method converges for any value of δ less than about 20.7 degrees, whereas Chew's algorithm converges for any $\delta < 28.6$ degrees.

Unfortunately, the non-acute domain restriction is too harsh, making these algorithms impractical in real-world scenarios. That is why variants were proposed that converge with acute angles too [Shewchuk 2000]. The basic idea of these variants is to avoid refining a triangle if it adjoins an acute angle in the input polygon. As a result, even if these algorithms converge, the meshes produced may still contain bad-quality triangles. Note that in [Shewchuk 2000], the size of the remaining bad triangles cannot be bound, meaning that simply removing them from the result may produce too coarse approximations.

Variants of Delaunay Refinement. Ruppert's original algorithm inspired several extensions. Shewchuk's `Triangle` software implements a variant that converges for angles up to about 26.5 degrees [Shewchuk 2002]. In [Miller et al. 2005], termination is guaranteed for non-acute domains and any $\delta < 26.45$ degrees. In [Pav and Walkington 2005], the method was extended to cope with curved surfaces. Extensions also exist for the 3D case [Shewchuk 1998]. The idea is very similar, though it requires that each input face is triangulated by a local 2D Delaunay mesh. These Delaunay triangles may be *encroached* if their diametral spheres contain a vertex other than the three triangle vertices. 3D Delaunay refinement proceeds by iteratively splitting all the encroached edges, then all the encroached triangles, and, finally, all the bad tetrahedra. Edges are split at their midpoint. Any encroached triangle is *split* by inserting its circumcenter c unless c encroaches upon one of the boundary edges. In that case, that edge is split instead. For each *bad* tetrahedron, its circumcenter c is inserted unless it encroaches upon one of the edges or one of the Delaunay triangles of the boundary. In that case, the edge or triangle is split instead. This process has similar requirements and guarantees as the 2D version. The input domain must not have acute angles (including dihedral angles) and termination is guaranteed up to a maximum value of δ representing the minimum angle of any triangle in the resulting tetrahedral mesh.

Corner-lopping. Corner-lopping was proposed by both Ruppert [Ruppert 1995] and Bern and colleagues [Bern et al. 1990] to restrict the input domain by creating holes around acute angles before the optimization, and by filling the holes by inserting the former acute angle apex and creating a fan of triangles out of it after optimization (see inset). One may think that, by using this approach, eventual small angles in the optimized mesh are at least as large as those in the input but, unfortunately, that is not the case, and smaller angles may appear [Shewchuk 2000] (e.g., because the hole boundary was subdivided during refinement). Furthermore, corner lopping is hardly extensible to 3D. In 2D, a circle is centered on each acute angle, and all incident segments are cut at their intersection with the circle. The acute vertex and the segment portions within the circle are deleted, thus leaving a hole, and the segment portions out of the circle are connected to each other by new segments that form a piecewise-linear approximation of the circle. In a trivial 3D extension one may think of using a sphere to similarly cut incident faces, and the portions remaining out of the sphere are connected by other faces that form a piecewise-linear approximation of the sphere. The problem is that it is unclear how to force these new faces to be free of acute angles.



Acute Angles. 3D domains with acute angles are strikingly more difficult to handle and, to the best of our knowledge, no robust implementation exists. In [Murphy et al. 2001], buffer zones around all boundary edges are tetrahedrized with prescribed patterns, whereas Chew's algorithm is used out of these zones. [Cohen-Steiner et al. 2002] creates a union of protecting balls near boundary edges and uses Ruppert's algorithm elsewhere. Buffer zones are used in [Cheng and Poon 2003] too but, differently from Murphy's work, their size adapts to the local feature size. Similar protecting regions are built in [Cheng et al. 2004] and [Rand and Walkington 2009] too, though the former defines the regions only implicitly, whereas the latter builds them explicitly and provides one of the first practical implementations. The state of the art is still represented by the method in [Shewchuk and Si 2014] that extends Chew's approach to 3D. This method is implemented in the famous `TetGEN` software [Si 2015].

2.3 Robust Implementations

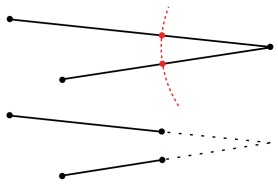
When implementing meshing algorithms, combining speed and robustness is particularly tricky. Delaunay and constrained Delaunay triangulations can be computed robustly by employing filtered geometric predicates [Shewchuk 1997] but, as soon as Steiner points are constructed along the process, these predicates are no longer

usable. In [Diazzi et al. 2023], Steiner points are represented implicitly, and the algorithm uses indirect predicates [Attene 2020b] to robustly create a CDT. To the best of our knowledge, the only existing implementation of a Delaunay refinement method which is robust enough for practical use is TetGEN [Si 2015]. TetGEN uses tolerances to consider too close points to be coincident or nearly coplanar points to be actually coplanar [Si and Shewchuk 2014]. In our tests (Sect. 8), TetGEN fails on 57 % of the cases if the tolerance is set to zero, and on 37 % of them if the default tolerance is used (10^{-8}).

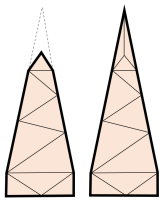
3 CHAMFERING AND MESHING IN 2D

We start describing our contributions by introducing the chamfering-based meshing in the easier 2D case. As mentioned in Sect. 2, variants of Delaunay refinement algorithms successfully work on domains with acute angles, though no guarantee can be given about the quality of resulting triangles near these angles.

Chamfering. Instead of leaving acute triangles untouched, our idea is to set a maximum acceptable distortion ϵ near acute angles and modify the input so that Ruppert's algorithm is guaranteed to converge. In 2D this amounts to simply *cut* each acute angle by removing a portion of its incident edges: we use the angle tip as center of a circle whose radius is the user-defined maximum distortion, and delete the edge portions that are contained in the circle. The two edges forming the acute angle are slightly shortened, and a little gap replaces the acute angle (see inset). Note that, differently from the corner lopping discussed in Sect. 2, we **do not** connect the cutting points.



Optimization. On such a modified input, we run Ruppert's optimization (which is now guaranteed to converge) and classify each triangle as *internal* only if it is entirely contained in the original input polygon, whereas it is *external* otherwise. An overview of the process is depicted in Fig. 2. This intermediate mesh M has graded density, has only good triangles, and conforms to the input everywhere but near the acute angles, where it can deviate up to ϵ. To get an exactly conformal triangulation, we enrich the original unchamfered input with all the vertices in M, split segments at their possible crossings with such vertices, and calculate a constrained Delaunay triangulation M'. By construction, M' is identical to M except near acute angles, where constrained triangles fill the gaps (see inset) and may not satisfy the quality requirements.



In the following section 4 we extend this construction to 3D and show how to overcome all the complications due to the additional dimension.

4 CHAMFERING 3D POLYHEDRA

In 3D we deal with polyhedra bounded by vertices, edges and planar faces. During the meshing process, edges and faces are called *constraints*.

Projection Condition. If the polyhedral surface satisfies the so-called *projection condition*, Delaunay refinement always converges for small enough values of δ [Shewchuk 1998]. The projection condition is formulated as follows: for any pair $< \sigma, \tau >$ of different constraints sharing at least a vertex, the projection of σ onto τ does not intersect the interior of τ, and vice-versa. This is equivalent as saying that none of the following angles is acute:

- the angle formed by two incident edges;
- the (dihedral) angle formed by two adjacent faces;
- the angle formed by an edge and a face sharing a vertex.

The goal of our chamfering process is to turn a generic polyhedron into a collection of polygons where the projection condition is satisfied.

Chamfering in 3D. In the remainder, we say that an edge is *acute* if its two incident faces form an acute angle or, equivalently, if their normals form an obtuse angle. We also say that a vertex is *acute* if one of the following conditions holds:

- one of its incident edges is acute;
- two of its incident edges form an acute angle;
- one of its incident edges forms an acute angle with its projection onto one of the incident faces.

The idea is to remove portions of the input surface around acute vertices and edges up to a maximum distance $\varepsilon > 0$. Differently from the 2D case, the eroded surface has a piecewise linear 1-dimensional boundary made of new edges that, if not properly created, may form new acute angles. The challenge is guaranteeing that it will not occur.

To better convey our intuition we first describe a direct extension of the 2D approach that, if naively used, might indeed produce new invalid angles. Then we explain how to properly modify the naïve approach to create a correctly chamfered polyhedral surface with no acute angles. From now on we assume that input faces are triangular (non triangular faces can be triangulated if necessary) and say that an edge is *flat* if its two incident triangular faces are coplanar. Flat edges are ignored when characterizing acute vertices though they are still subject to chamfering.

4.1 Naïve Chamfering

A trivial extension of the 2D chamfering in Sect. 3 to three dimensions uses spheres to cut constraints incident at acute vertices and cylinders to cut constraints incident at acute edges. To keep the construction simple we need that at least a portion of each triangle remains after the cut. This is ensured if the sphere around a vertex does not contain any constraint but those directly incident at the vertex. Similarly for cylinders around edges. Shewchuk's definition of the local feature size (lfs) [Shewchuk 2000] may be used for this purpose. For any input vertex v, the local feature size $\mathbf{lfs}(v)$ is the distance from v to the nearest input vertex, edge or face that is not incident to v. For any point p lying on an input edge e, $\mathbf{lfs}(p)$ is the distance from p to the nearest input vertex, edge or face that is not incident to e. $\mathbf{lfs}(e)$ is the minimum **lfs** of all the points on e.

Local LFS. Using the local feature size to bound the cutting distance ensures that each of the input triangles is never cut too much,

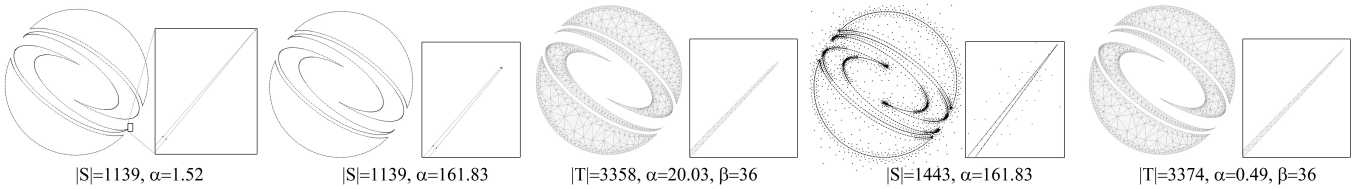


Fig. 2. Delaunay refinement with chamfering. The input model (left) was chamfered to remove acute angles (mid-left) and then Delaunay-refined to produce our intermediate mesh (center). Vertices of the intermediate mesh along with all the segments (mid-right) formed the input to the final enriched CDT (right). In the image, |S| is the number of segments, |T| is the number of triangles, α is the absolute minimum angle, and β is the average minimum angle across all the triangles.

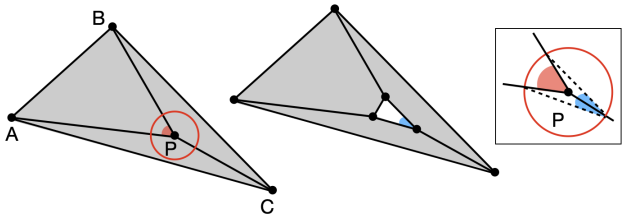


Fig. 3. The tetrahedron on the left has a vertex P very close to its opposite face ABC. The red angle is acute and consequently P is an acute vertex. By removing P using the naïve chamfering procedure a new acute angle is introduced (right).

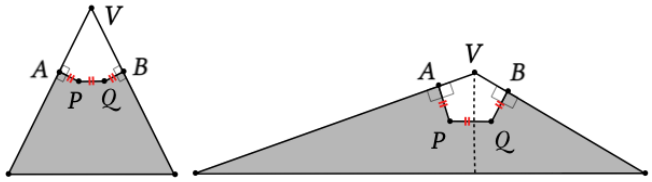


Fig. 4. Chamfering a triangle around an acute vertex. When the chamfering distance is 1/3 of the LFS at V (the height of the triangle at V in these examples), and because bridge edges have the same length, points P and Q are inside the triangle by construction.

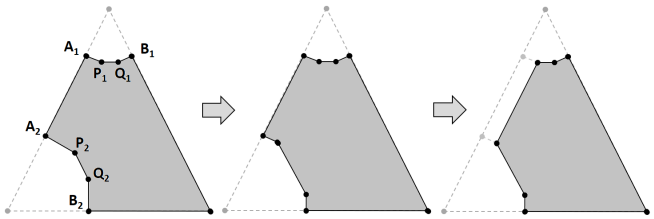


Fig. 5. Chamfering an acute edge $< A_1, A_2 >$. P_2 is moved towards A_2 so that their distance is equal to the distance between A_1 and P_1. Similarly, Q_2 moves towards B_2. After this modification, the quadrilateral $< A_1, A_2, P_2, P_1 >$ becomes a rectangle which is removed to chamfer the acute edge.

meaning that a triangle never disappears or changes topology. However, the same objective may be obtained using a *local* version of the LFS which is much simpler to compute. For the sake of our chamfering algorithm, the local LFS at a vertex v (or edge e) is simply the smallest distance of v (or e) from its *link* [Attene et al. 2009].

Chamfering. For each acute vertex v we split all its incident edges at a distance $d = \min(\varepsilon, \textbf{local\_lfs}(v)/3)$. Then we subdivide each incident face by connecting the two split points on its edges meeting at v. Finally, we remove the triangular portions of these faces which are still incident at v, thus leaving a hole. Then, for each acute edge e we consider all its incident faces and, from each, we remove the quadrilateral portion inside the cylinder of radius $d = \min(\varepsilon, \textbf{local\_lfs}(e)/3)$ and axis e.

Failure example. Imagine the following construction illustrated in Fig. 3. Split a triangle into three sub-triangles and slightly lift the split point (let it be P) over the plane of the triangle. P is an acute vertex that, if processed as just described, leads to a triangular hole whose bounding edges still form acute angles.

4.2 Orthogonal Chamfering

To avoid the creation of new acute angles we modify the naïve approach as follows.

Vertex Chamfering. Let t be a triangle with an acute vertex V. We still split its two edges incident at V but, instead of simply connecting the two new points, we create a *bridge* made of three new edges. With reference to Fig. 4-left, the two points A and B were created by splitting the edges incident at V as described in Sect 4.1. The points P and Q are selected on the edge-orthogonal straight lines by A and B respectively, so that the three segments AP, PQ and QB have the same length. Finally, we remove the pentagon that remains incident at V. Note that a triangle t may be chamfered at V (Fig. 4-right) even if it is obtuse at that vertex. This happens when another triangle t' incident at V is acute at V. In this case d is necessarily smaller than $h/3$, where h is the triangle height at V, and this guarantees that points P and Q are inside triangle.

Edge Chamfering. When all the acute vertices are removed we proceed with chamfering possible acute edges. First, we observe that any face having an acute edge is no longer a triangle because of the vertex chamfering (Fig. 5-left). Furthermore, the endpoints of an acute edge are two newly inserted points (A_1 and A_2 in Fig. 5), each connected to one of the two internal points (P_1 and P_2) of the two bridges. We proceed as follows: if a face has acute edges, we consider all the points which were inserted on the edges of the initial triangle. We call them LNCs (for reasons which become clear in the next section): A_1, A_2, B_1 and B_2 in Fig. 5, can be up to

six. Then, we consider all their distances from the internal points they are connected with, pick the minimum distance d, and move all the internal points towards their closest LNC so that all their distances become equal to d (Fig. 5-center). Finally, for each acute edge e and for each of its incident faces f, the endpoints of e and their corresponding internal points on f form a rectangle which is removed from f (Fig. 5-right). Besides avoiding the creation of new acute angles (a formal proof is reported in Appendix B), the idea of this construction is preventing the production of unnecessarily short edges or small local feature sizes.

4.3 Robust implementation

When the chamfered polyhedron is processed by Delaunay refinement, numerical robustness is critical. We introduce a novel robust and efficient approach that avoids the problems usually introduced by numerical rounding by using a strategy inspired by [Diazzi et al. 2023]: we represent all the newly inserted points implicitly so that indirect predicates [Attene 2020b] always give the correct result.

We observe that our new points are either on input edges (e.g. A and B in Fig. 4) or in the interior of input triangles (P and Q). In [Diazzi et al. 2023], implicit points called LNCs were defined to represent Steiner vertices. In our case, we can use LNCs to represent the first type of Steiner points on input edges. Unfortunately, no existing implicit point type is appropriate to represent the second group of points we insert in the interior of triangles. Therefore, we introduce a new construction for these implicit points.

In the following subsection, we briefly summarize what an LNC is and how it works in practice, introduce the new point type we need, and then show how to use them in the predicates needed by our algorithm.

4.3.1 LNCs and BPTs. An LNC (LiNear Combination) point is an object formed by two different 3D points $\mathbf{p}$ and $\mathbf{q}$ and a value t s.t. $0 \leq t \leq 1$, and corresponds to the 3D point $t\mathbf{p} + (1-t)\mathbf{q}$.

This concept is easily extended to points on triangles as follows. A Barycentric Point on Triangle, or BPT for short, is an object made of three explicit, different and non-aligned points $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$ and two barycentric coordinates u and v, such that:

$$0 \leq u \leq 1, 0 \leq v \leq 1, u+v \leq 1. \tag{1}$$

Its position in space is $u\mathbf{p}+v\mathbf{q}+(1-u-v)\mathbf{r}$. Each of the three points $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$ is represented explicitly by its three Cartesian coordinates that, in turn, are encoded by floating point (FP) numbers as u and v.

When computing the coordinates of a BPT using FP arithmetic, numerical rounding may move the point out of the triangle plane, and subsequent calls to geometric predicates, even if *exact* [Shewchuk 1997], may provide unexpected results and put the program in an inconsistent state. Conversely, an indirect predicate combines the expression of the BPT (or any other implicit point) with the expression of the predicate, so that the overall expression is a function of exact values and can be robustly evaluated.

Conversion. The conversion of a point $\mathbf{w}$ expressed in Cartesian coordinates to a BPT on a triangle $\mathbf{p}$,$\mathbf{q}$,$\mathbf{r}$ amounts to deriving the values of u and v as follows:

$$u = \Delta(\mathbf{w},\mathbf{q},\mathbf{r})/\Delta(\mathbf{p},\mathbf{q},\mathbf{r}),$$
$$v = \Delta(\mathbf{w},\mathbf{r},\mathbf{p})/\Delta(\mathbf{p},\mathbf{q},\mathbf{r}),$$

where Δ represents the triangle area. These values can be computed using FP arithmetic with a minimal care to ensure that Equation 1 holds: if u is slightly larger than 1 due to rounding we snap u to 1 and v to zero; same for v; if $u+v>1$ we compute $d = u+v$ while rounding towards $+\infty$, then divide both u and v by d while rounding towards $-\infty$. A so-computed BPT might not be exactly coincident with the original $\mathbf{w}$, but it is exactly on the triangle and, for the purposes of Delaunay refinement, this is sufficient. Note that, in principle, rounding might make angles at A and B slightly acute. Although it would be possible to control the rounding so that this does not happen, Delaunay refinement keeps converging even if angles are as small as 60 degrees [Shewchuk 2002]. Therefore we argue that this is not an issue in practice. Indeed, even without such an additional control, our implementation succeeds on **all** the models in our dataset (Sect. 8). Finally, we point out that in our method $\mathbf{p}$, $\mathbf{q}$ and $\mathbf{r}$ are never aligned because they are vertices of a valid input triangle.

4.3.2 Checking the Projection Condition. A robust implementation requires predicates to discover whether an angle is acute or not. The angle formed by three points $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$ is acute if the sign of the dot product $(\mathbf{p}-\mathbf{q})(\mathbf{r}-\mathbf{q})$ is positive, which can be robustly evaluated by our predicate `dotProductSign3D(p,q,r)` (Appendix A.1) using filtered FP arithmetic.

Similarly, the dihedral angle formed by two triangles $\mathbf{a}$, $\mathbf{c}$, $\mathbf{d}$ and $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ on their common edge $\mathbf{c}$, $\mathbf{d}$ is acute if the sign of the following dot product is positive:

$$[(\mathbf{b}-\mathbf{d})\times(\mathbf{c}-\mathbf{d})]\,[(\mathbf{a}-\mathbf{d})\times(\mathbf{c}-\mathbf{d})] \tag{2}$$

where $\times$ indicates the cross product. Our predicate `dihedralSign(a,b,c,d)` (Appendix A.2) robustly calculates the sign of Eqn. 2.

Finally, the projection of an edge $\mathbf{a}$,$\mathbf{q}$ on a triangle $\mathbf{a}$,$\mathbf{b}$,$\mathbf{c}$ intersects the interior of the triangle if the dihedral angle formed by $\mathbf{q}$ and $\mathbf{c}$ on $\mathbf{a}$, $\mathbf{b}$ is acute and the dihedral angle formed by $\mathbf{q}$ and $\mathbf{b}$ on $\mathbf{a}$, $\mathbf{c}$ is acute. Namely, if both `dihedralSign(q,c,a,b)` and `dihedralSign(q,b,a,c)` are positive.

The exact calculation of these predicates along with their floating point filters are described in Appendix A.

4.4 Simplified Chamfering

The orthogonal chamfering discussed in Section 4.2 is simple and, by construction, it conservatively guarantees that no new acute angles are introduced during the process. Nonetheless, in many cases it creates more vertices than necessary and the three-edge *bridges* can be reduced to two or even just one edge without introducing new acute angles, while increasing the local edge length (See Fig. 6).

Bridge Removal. We introduce a conservative algorithm that removes unnecessary bridges. First, all bridges are ordered according to the increasing length of their shortest edge. Then, starting from the first bridge, a temporary edge is created by using the extreme endpoints of the bridge and angles formed with the connected

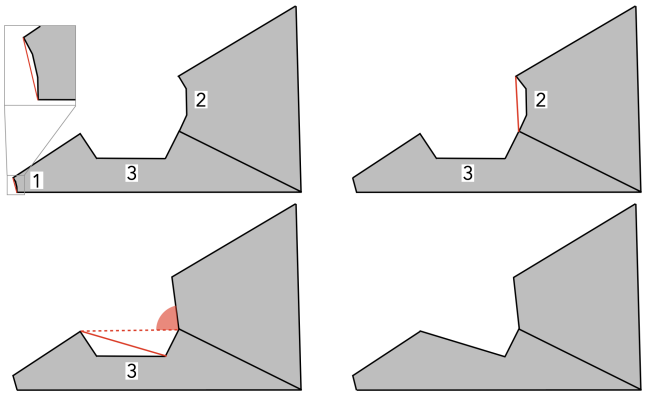


Fig. 6. An example of bridge simplification. There are 3 bridges, each made of 3 edges, numbered from 1 to 3 according to the ascending shortest edge in the bridge. Bridge nr. 1 is simplified first (up-left) and replaced with the red candidate shown in the zoom. Then, bridge nr. 2 is simplified (up-right). During the simplification of bridge nr. 3 (down-left) the dotted red candidate is tried first, but is rejected to avoid the introduction of the red acute angle; the solid red candidate is chosen instead and the final result is obtained (down-right).

boundary edges and the incident faces are measured: if none of them is acute the temporary edge replaces the bridge, otherwise it is rejected. Upon rejection, the two possible simplifications involving left-central and right-central edges are tried (shortest bridge-edge first) and rejected in case of new acute angles.

Evaluation. Our experiments, which were run on thousands of models (Sect. 8), confirm that this simplification often leads to lower computational costs, significant reduction of the number of tetrahedra and increased shortest-edge length in the refined mesh. Indeed, longer input edges drastically reduce the need for refinement in the subsequent optimization phase.

Using a unique value of ε is not mandatory. In practice, when exact conformity is required, ε can vary across the mesh as the surface portions that we chamfer out are eventually reintegrated (Sect. 6). In order to maximize the edge lengths produced by the chamfering, and consequently reducing the number of tetrahedra produced by the whole algorithm, in our experiments we use a local adaptive ε: for each chamfered vertex v we set ε_v to one third of **local_lfs**(v) (see Sect. 4).

5 MESHING THE CHAMFERED POLYHEDRON

The result of the chamfering represents the input to our 3D Delaunay refinement. The overall process is summarized in Algorithm 1, whereas specific phases are described in the remainder of this section.

5.1 Outer box

Before starting the meshing we create six rectangular faces, each made of two coplanar triangles, forming a box around the chamfered mesh, and enrich the input with them. During the process we make sure that none of the 2D Delaunay triangles is encroached (see sect 5.3), including those on the outer box. This guarantees that no circumcenter can be out of the box.

5.2 DT and Segment Recovery

Let P' be the so-enriched chamfered polyhedron. We create a Delaunay tetrahedrization M of the vertices in P', and then iteratively split all the encroached segments in P' as mentioned in Sect. 2. Upon convergence, each segment in P' coincides with a straight chain of edges in M, and each face in P' has a refined boundary.

5.3 Local Delaunay and Face Recovery

Now we compute a local 2D Delaunay triangulation of each of the faces. Each of these Delaunay triangles is searched in M: if it is either missing or encroached, we split it by inserting its circumcenter unless this circumcenter encroaches one of the boundary edges. In the latter case, we split the encroached edge instead. Upon convergence, each input face coincides with a set of coplanar triangles in M.

5.4 Optimizing Tetrahedra

We create a priority queue of all the *bad* (according to δ) tetrahedra sorted by the ratio $B = r/s$ of the circumradius r over the shortest edge length s. At each step, we pick the first tetrahedron in the queue (which has the largest B) and split it as described in Sect. 2. We update the queue to account for the new possibly bad tetrahedra. When the queue is empty all the tetrahedra have their smallest face angle larger or equal to δ.

5.5 Interior Characterization

In this last phase, we select the tetrahedra in M that are entirely contained in P. Note that P is the unchamfered version of the input, meaning that near acute angles some tetrahedra in M may be only partly inside and thus not selected.

ALGORITHM 1: Delaunay Refinement

Require: A closed, manifold and oriented triangulated surface mesh P bounding a polyhedron;
Require: The chamfered version P' of P;
Require: Minimum target angle δ.
Ensure: A quality tetrahedral mesh M which conforms to P everywhere but near acute angles in P.

```
1: P' := Add outer box;                                   ▷ Sect. 5.1
2: M := Delaunay Tetrahedrization of P' vertices;         ▷ Sect. 5.2
3: Recover_Segments(P', M);                               ▷ Sect. 5.2
       for each face f in P' do
   └   Create local 2D Delaunay Triangulation of f;
4: Recover_Faces(P', M);                                  ▷ Sect. 5.3
5: Optimize_Tetrahedra(P', M, δ);                         ▷ Sect. 5.4
6: Mark_Internal_Tets(M, P);                              ▷ Sect. 5.5
```

5.6 Robustness Issues

As mentioned, we use LNCs and BPTs to represent split points on edges and triangles respectively during the chamfering. Furthermore, we keep using these two point types when creating Steiner points on segments and faces during refinement. We observe that Steiner points in the volume do not need to be exactly represented:

given a bad tetrahedron t, any point in its circumsphere is appropriate to split t, and we pick the circumcenter just to maximize the length of the new edges being produced. Therefore we use the rounded floating point circumcenter in these cases. Note that for the rounded point to jump out of the sphere, the sphere radius should be as small as machine precision, which is only possible if the tetrahedron is itself as small as machine precision. That is an extreme case that we have never experienced in our large-scale tests (Sect. 8). Due to the grading, such a small tetrahedron would require a massive number of other tetrahedra in its neighborhood and we argue that all the computational resources would be consumed far before reaching that point.

5.7 Avoiding Slivers

Algorithm 1 converges for any $\delta < 20.7$ degrees [Shewchuk 1998] and produces tetrahedral meshes where no face has angles smaller than δ. However, dihedral angles can be arbitrarily small in slivers, which are nearly flat tetrahedra having no extreme face angles. To avoid their creation during the optimization phase (Sect. 5.4) we calculate the shortest edge length e and the shortest distance h between two opposite edges in each tetrahedron, and declare the tetrahedron to be *bad* if the ratio e/h is larger than a constant ρ ($\rho = 7.2$ in our tests). This modification is effective and does not prevent convergence in practice, though we could not formally demonstrate it. That is why we have left it as an optional feature in our prototype code.

6 CONFORMING TO THE INPUT EVERYWHERE: ENRICHED CDT

Our last step enables our algorithm to exactly preserve the input surface at the cost of losing its angle guarantees but still retaining a good average quality and algorithmic robustness.

After running Algorithm 1, we form a new collection of vertices and polygons by merging elements from the optimized mesh M and the input polyhedral surface P. With reference to Fig. 7 we include:

- all the vertices in M except those in the chamfering spheres and cylinders (Fig. 7-b);
- all the vertices in the original unchamfered input P (Fig. 7-c). Possible duplications with the previous vertices are removed;
- all the Delaunay triangles that partition faces in the subdivided P' (Fig. 7-d);
- the original face portions that were cut out by the initial chamfering, possibly subdivided by inserting bounding vertices from P' (Fig. 7-e).

This construction is a polygonal mesh surface surrounded by additional points, and the geometric realization of the surface exactly coincides with P. We then process this collection using a classical Constrained Delaunay Tetrahedrization (CDT) algorithm (we used the implementation of [Diazzi et al. 2023]). The CDT of this collection is, by construction, identical to M but near the acute angles where possible bad tetrahedra may appear.

Note that vertices in M might be implicit and hence not directly usable as input to [Diazzi et al. 2023]. Furthermore, using their rounded floating point coordinates is not an option as it may lead to invalid configurations that may cause a failure of [Diazzi et al. 2023]. To achieve full robustness we use the recently introduced *Cascaded Implicit Points* feature in the Indirect Predicates library [Attene 2020a] to implement the CDT algorithm with support for input in both explicit and implicit form. Thanks to that we could slightly modify the original CDT algorithm [Diazzi et al. 2023] to make it support input in both explicit and implicit form (see Appendix C).

ALGORITHM 2: Delmesher

Require: A closed, manifold and oriented triangulated surface mesh P bounding a polyhedron;

Ensure: A tetrahedral mesh M which conforms to P, whose tetrahedra satisfy quality guarantees everywhere but near acute angles in P.

```
1: P' := Chamfer(P);                          ▷ Sect. 4
2: M' := Delaunay Refinement of P';           ▷ Algorithm 1
3: M := Enriched_CDT(M', P', P);              ▷ Sect. 6
```

7 INPUT FILTERING

When using good-quality elements to mesh a given domain, an extremely small LFS may require a lot of such elements. For example, in the simple rectangle meshed by quadrilaterals in the inset the number of elements doubles as the height halves. In 3D this is even worse: in a box meshed by hexahedra the number of elements increases by a factor of four as the height halves. In general a too small LFS may easily make the algorithm consume all the computational resources or produce too complex meshes that lose their practical utility. A concrete example is shown in Fig.8. Here three very similar input models are considered: they have the same shape but, from top to bottom, the thickness h of one of their sides becomes smaller and smaller. As expected, while h decreases, the number of tetrahedra tends to explode, and so does the request for computational resources. Note that tetrahedron density increases only in the region where the LFS is actually reduced.

Even in the unlikely circumstance where users are willing to wait for their creation, meshes made of trillions of elements are hardly usable in most practical applications. Therefore we provide an input filtering strategy to trade off the output complexity with the mesh quality, while still preserving 100% conformance. Specifically, we plainly remove all the input triangles that are locally responsible for a small LFS. We set a user-controllable tolerance to ensure that the minimum LFS of the filtered PLC is large enough to avoid an explosion. Removed constraints are reintroduced during the enriched CDT phase (Sect. 6) with a strategy similar to the one adopted for the chamfered regions. The filtering procedure relies on two different steps: the first is directly related to the input triangulation, while the second is iteratively alternated to chamfering.

Filtering first step. Given the input tolerance τ, we remove from the input surface P all the triangles responsible for an LFS smaller than τ. To efficiently evaluate the distance between disjoint simplexes we compute the CDT of the original input surface and navigate its tetrahedra.

Filtering second step. When chamfering is executed on the filtered input $P^{(0)}$, edges shorter than τ and other kind of small LFS

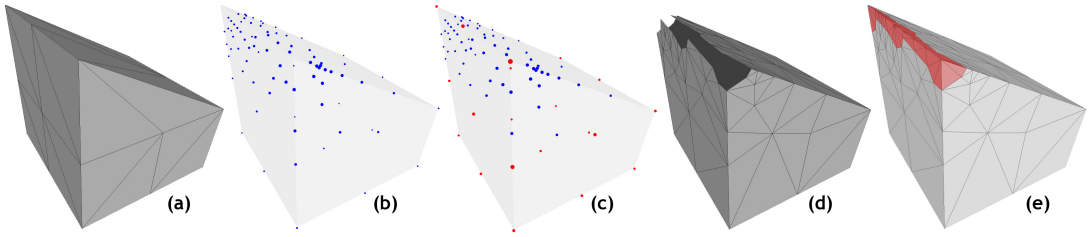


Fig. 7. Creation of the input for the enriched CDT. Input polyhedron P (a); vertices of the refined mesh M out of the chamfered regions (b); vertices of P (c, red dots); Delaunay triangles of P' (d); chamfered portions of P (e).

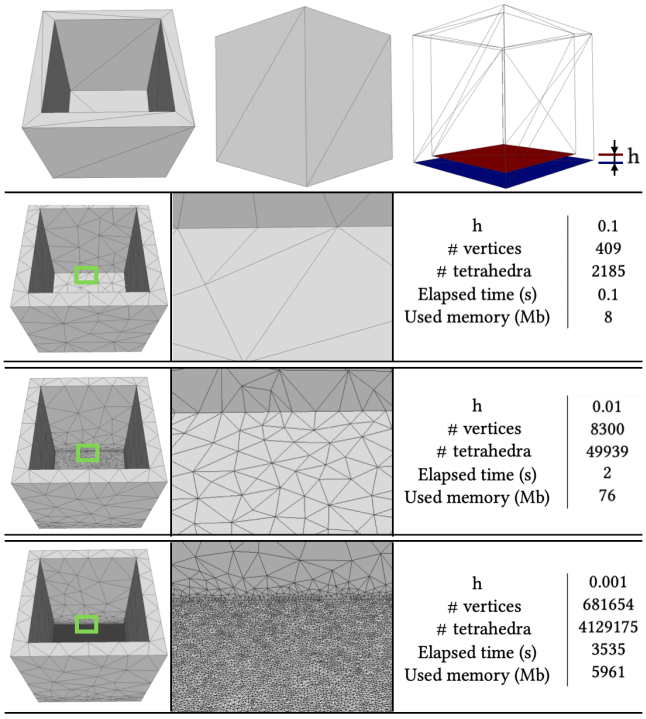


Fig. 8. In the input surface (first row) the distance between the red and the blue faces is indicated by h. The surfaces obtained by choosing $h = 0.1$, 0.01 and 0.001 are then meshed by our algorithm and results are reported in the second, third and fourth row respectively. Note that the number of elements and the request for computational resources grows much faster than $1/h$, and this is unavoidable because elements are required to have a good quality.

configurations may be reintroduced. Thanks to the local effect of the chamfering operations, identifying these undesired configurations is easy and does not require to recompute a new CDT. If a simplex s violates the LFS bound we do not remove it directly: instead, we remove from $P^{(0)}$ all the triangles that are responsible for the introduction of s during chamfering. This operation turns $P^{(0)}$ to a new filtered version $P^{(1)}$. We then iteratively repeat this filtering+chamfering sequence $P^{(n)}$ until no more undesired configurations appear, and use the last chamfered surface as input to Algorithm 1. Termination is ensured because at each iteration we remove at least an input triangle, meaning that the number of iterations is bounded by the number of triangles in P. In the extreme case where *all* the triangles are removed, the Delaunay refinement uses only the outer box triangles (see Sect. 5.1), while all the constraints will be reintroduced during the enriched CDT step.

As shown in Sect. 8 this filtering strategy makes our algorithm able to successfully mesh all the models in our dataset while consuming a reasonable amount of computational resources.

8 RESULTS AND DISCUSSION

We have developed a prototype implementation of our algorithm in C++, using and extending upon the publicly available indirect predicates library [Attene 2020a]. All the experiments were run on a Linux-based machine with AMD EPYC 7452 CPU and 1Tb RAM. All the experiments were run on a single core. Our reference dataset was obtained by selecting the 3942 valid models from the Thingi10k dataset [Zhou and Jacobson 2016]: a mesh is *valid* if it is closed, manifold, oriented, and without self-intersections. Within this set of models, the minimum LFS is 3.15×10^{-18} times the bounding box diagonal, which is far too small to obtain usable meshes in a reasonable time. Hence, in our experiments we use a tolerance τ as described in Sect. 7.

8.1 Experiments and Results

We used an input filtering tolerance $\tau = 10^{-4}$ times the bb-diagonal, and set $\delta = 14.47$ degrees. Note that the value of δ corresponds to a maximum ratio $B = 2$ of the circumradius over the shortest edge length in a tetrahedron, the same default value used by TetGEN.

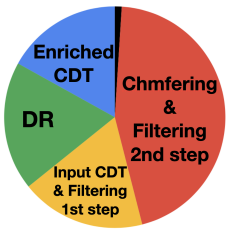
We perform two different experiments. The first, called *Safe-Delmesher-e4*, does not attempt to remove slivers, whereas the second, *Delmesher-e4*, does as described in Sect. 5.7 to produce better shaped tetrahedra. Even if formal convergence guarantees are lost in the second version, we did not experiment any failure during our tests. Both experiments succeeded on *all* models of the dataset and over the entire set of output meshes we measured the quantities listed in Table 1.

All the angles are measured as robustly as possible as described in [Shewchuk 2013]. In both experiments, the time taken by our algorithm to converge (see time distribution in inset image) is dominated by the iterative chamfering+filtering (Sect. 7, second step, red), then by the input CDT computation and filtering (Sect. 7,

Table 1. Average measurements for experiments *Safe-Delmesher-e4* and *Delmesher-e4* on the dataset of 3942 Thingi10k valid models. For all the rows the average is taken with respect to the entire dataset. *Avg. min* indicates that the absolute minimum is computed on each single model and the average of all these minima is reported. Similarly for *Avg. max*. Values in brackets indicate angle values at the end of Alg. 1 but before the final *enriched CDT* phase.

	Safe-Delmesher-e4	*Delmesher-e4*
Avg. # tetrahedra	590725	669177
Avg. # vertices	106215	118878
Avg. min face angle	2.25 (14.57)	2.24 (14.58)
Avg. max face angle	172.78 (148.51)	172.88 (148.13)
Avg. min dihedral angle	0.09 (0.13)	0.69 (4.22)
Avg. max dihedral angle	179.79 (179.7)	177.92 (166.56)
Absolute min face angle	0 (14.48)	0 (14.48)
Average time (s)	77	82
Average memory (Mb)	637	727

first step, yellow), the Delaunay refinement (green) and the enriched CDT (blue), whereas all the other steps (black) are negligible. Even if one might expect that Delaunay-refinement is dominant, the chamfering+filtering phase is usually repeated many times, especially when the input has many coplanar triangles. Conversely, filtering ensures that there are no "tiny regions" that Delaunay refinement has to fill with tons of small well-shaped tetrahedra, thus reducing the cost of the refinement phase.



On our dataset, experiment *Safe-Delmesher-e4* takes on average 125 μs per tet; convergence is reached in less than 1 minute for 16.23% of the models, between 1 and 10 minutes for 72.94%, between 10 and 60 minutes for 10.75%. For the remaining models (less than 0.08%) it terminates in less than 3 hours. We measured a very similar time distribution for experiment *Delmesher-e4* which takes on average 114 μs per tet, reaching convergence in less than 1 minute for 14.59% of the models, between 1 and 10 minutes for 73.26%, between 10 and 60 minutes for 12.07% and in less than 3 hours for the remaining models (0.08%). Only model 719791.off required about 36 hours to terminate for both experiments: we observe that on this same model the CDT alone requires more than 10 million Steiner points and the longest computation time even in [Diazzi et al. 2023], while TetGEN fails to process it. Fig. 9 depicts some of the meshes obtained through experiment *Delmesher-e4*. Note that Shewchuck proves [Shewchuk 2002] that Delaunay refinement converges because it never produces edges shorter than the shortest input LFS, and this lower bound can be used to estimate an upper bound on the number of tetrahedra (e.g. KV/L^3, where V is the total volume being meshed, L is the shortest edge length, and K is a small constant).

8.2 Comparison

When dealing with conformal quality meshing the most relevant state of the art is represented by TetGEN, whereas maximum robustness was obtained by CDT [Diazzi et al. 2023]. Hence we compare with these two existing works which are both able to create a conformal mesh of the convex hull of an input polyhedron. Unfortunately there is no common file format that can be parsed by all the implementations available for these previous works, and format conversion might introduce small deviations. Hence, to ensure a fair comparison, we parse the input files using the same function used by our method, then fill TetGEN's and CDT's data structures with the so-loaded data and call the available library functions (`tetrahedralize` and `createSteinerCDT` respectively) to run the algorithms on the exact same input.

We used the latest version of TetGEN (v1.6.0 released on August 2020) with options `zpcqT0`. These options make TetGEN create a conformal mesh (option `p`) of the convex hull of the input (option `c`) with tetrahedra whose circumradius over shortest edge ratio is bounded by 2 (option `q`) and without allowing any distortion of the input (option `T0`). With these options, TetGEN succeeds on 1701 models out of 3942 (43 %) and produces meshes with the angles reported in the second column of table 2. We call this experiment TetGEN-T0. If the option `T0` is not used TetGEN is allowed to move the input up to a maximum tolerance distance $T = 10^{-8}$. Since this is its default behaviour we have run it in this configuration too, and the success rate grows from 43 % to 63 %, while angles do not exhibit significant changes (see third column of table 2). In both cases, TetGEN takes 31 μs per tet on average (for the sake of comparison we recall here that *Safe-Delmesher-e4* and *Delmesher-e4* take on average 125 μs per tet and 114 μs per tet respectively).

To compare with [Diazzi et al. 2023] we used the public CDT code released by the authors. When used with standard options, this code produces a Constrained Delaunay Tetrahedrization of the convex hull with no approximation. CDT succeeds on all the models and produces meshes with the angles listed in fourth column of table 2. The CDT library takes 11 μs per tet on average.

Fig. 10 shows that extreme angles produced by *Safe-Delmesher-e4*, *Delmesher-e4*, *CDT*, *TetGEN-T0* and *TetGEN* are comparable, while from Fig. 11 it is possible to see that the average quality of *Safe-Delmesher-e4* and *Delmesher-e4* elements is higher than the other three methods. We note that this behavior may be emphasized by the larger number of tetrahedra produced by our methods.

As a particular example, Fig. 12 depicts differences when our method without filtering (referred as *Delmesher*) is compared with *Delmesher-e4* and TetGEN. The image depicts model 71331 in the Thingi10k dataset and details of the tetrahedral faces produced by the methods being compared. Measured quantities are collected in table 3. We observe that our algorithms produce a better distribution of high quality elements, as shown in Fig. 13, at the cost of a higher number of mesh elements. Interestingly enough, results of *Delmesher* and *Delmesher-e4* show a similar distribution, thus confirming that input filtering has a negligible impact on the eventual mesh quality. Furthermore, bad elements are concentrated around a few spots (i.e. where the input was chamfered), whereas good quality elements are

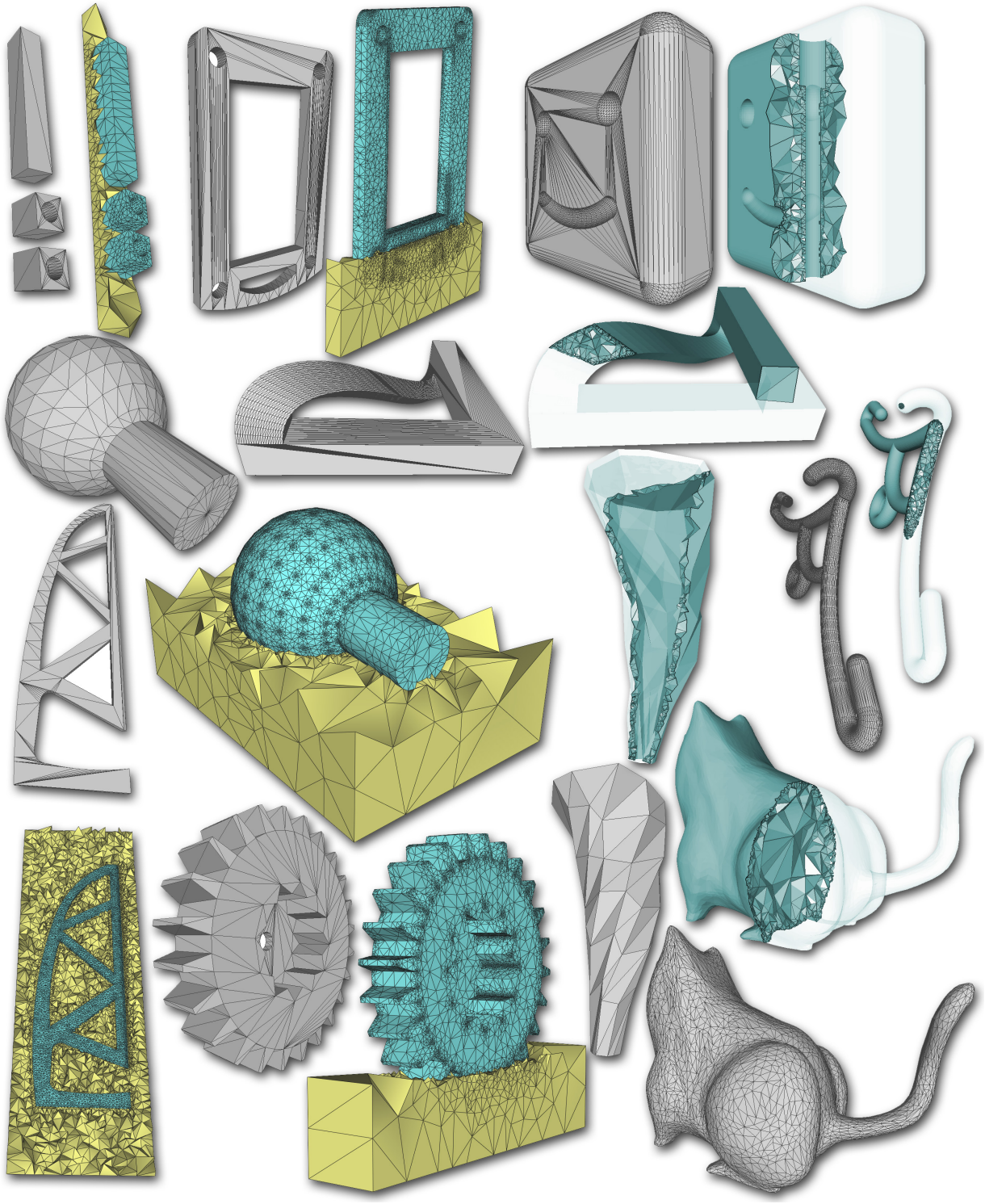


Fig. 9. Ten models in the Thingi10k dataset. The image depicts the original input models along with the meshes produced by our algorithm. On the left, meshes are depicted along with their containing box, whereas on the right the internal tetrahedra are shown.

Table 2. Average output parameters measured for TetGEN-T0, TetGEN, CDT, *Safe-Delmesher-e4* and *Delmesher-e4*. While CDT, *Safe-Delmesher-e4* and *Delmesher-e4* converge on the whole 3942 models of dataset, TetGEN and TetGEN-T0 do not. The reported averages were taken on the common subset of 1701 models on which both TetGEN and TetGEN-T0 succeed. Average min indicates that the absolute minimum is computed on each single model and the average of all these minima is reported. Similarly for Average max.

Measure	TetGEN-T0	TetGEN	CDT	*Safe-Delmesher-e4*	*Delmesher-e4*
Average min face angle	2.0	2.0	1.3	3.6	3.6
Average max face angle	171.4	171.1	172.1	168.4	168.6
Average min dihedral angle	1.2	1.2	1.0	0.2	1.2
Average max dihedral angle	177.0	117.0	175.8	179.6	176.6
Average out vertices	29870	29475	7049	31005	35671
Average out tetrahedra	176700	173950	49086	184339	213050

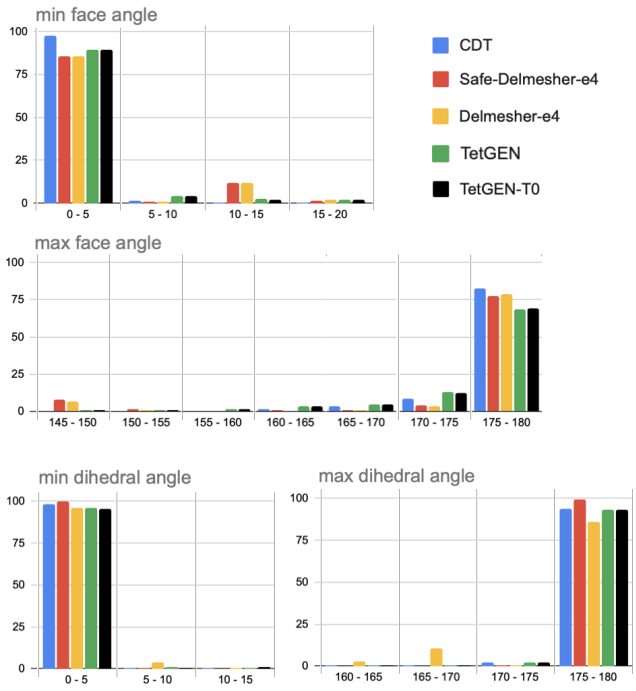


Fig. 10. Histograms showing the distribution of *extreme* angles among the meshes produced in our experiments. The height of each bar corresponds to the percentage of models having the angle, indicated by the title, in the range displayed at its bottom. For each experiment the percentage is wrt the total number of models on which that experiment succeeded.

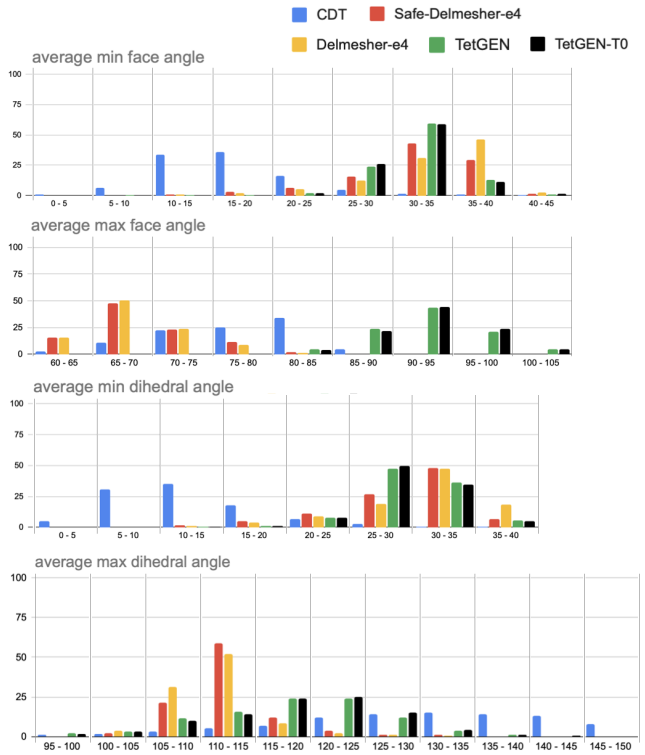


Fig. 11. Histograms showing the distributions of *average* angles among the meshes produced in our experiments. Here by *average* we consider an *average-by-model*. Specifically, the *average min face angle* of a given mesh is computed by summing the minimum angle of each face and then dividing by the total number of mesh faces. The same procedure involving maximum face angle is used to compute the *average max face angle*. Similarly, the average min/max dihedral angles of a given mesh are computed by summing the min/max dihedral angle of each tetrahedron and dividing by the number of tetrahedra. These histograms can be interpreted as in Fig. 10.

dominant. Though this is not surprising, a quantitative analysis of angle histograms provides a confirmation in practice (Fig. 13-right).

Similar results were obtained for all the other models shown in this paper (see the additional material). A dominance of good-quality elements is particularly important when the mesh is paired with solver strategies to decouple mesh quality from solution quality [Schneider et al. 2018]. Furthermore, our method is robust both in theory and practice. While *Delmesher-e4* always succeeds in a reasonable time, we point out that *Delmesher* is still guaranteed to converge though we cannot say how many tetrahedra it must create to do so.

Not surprisingly, our method significantly improves average angles when compared with [Diazzi et al. 2023], although both have a 100% success rate. Clearly, CDT produces lighter output meshes consuming less computational resources because it does not strive to obtain quality meshes and hence inserts much less Steiner points.

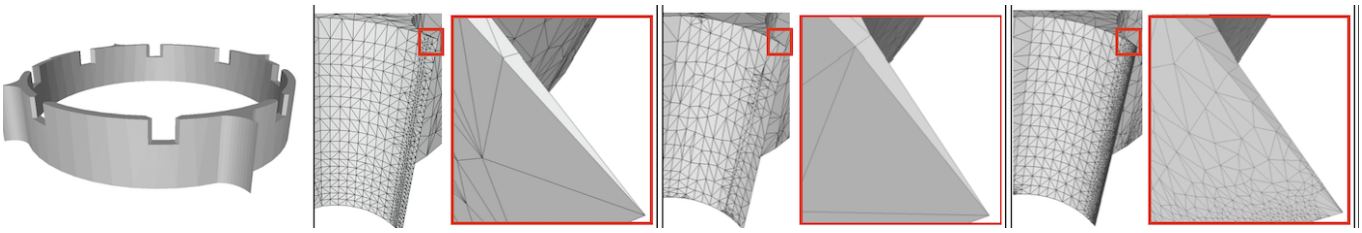


Fig. 12. An input model with very sharp angles (left) meshed by TetGEN (left-center) and by our method *Delmesher-e4* with filtering tolerance $\tau = 10^{-4}$ (left-right) and by *Delmesher* without filtering (right). All the produced meshes are conformal but, differently from the first two, *Delmesher* inserts a huge number of small tetrahedra to fill the tiny region highlighted in the zoom.

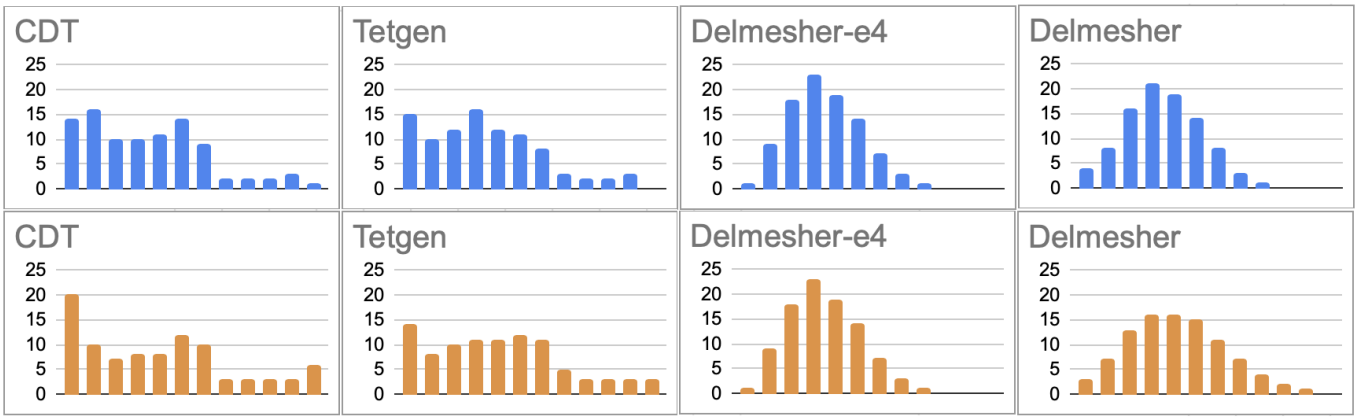


Fig. 13. Histograms of angles in meshes of model 71331.off produced by [Diazzi et al. 2023] (CDT), TetGEN, our method with filtering tolerance set to 10^{-4}, and our method without input filtering. In each histogram the range of angles (0,180) is split into 12 bins of 15 degrees each, and each bar represents the percentage of angles in that bin. We report both face angles (top row, blue) and dihedral angles (bottom row, yellow). Most face angles are in the range 15-30 degrees for CDT, whereas they are in 45-60 for all the other methods. Noticeably, Both CDT and TetGEN have their peak number of dihedral angles in the range 0-15, while our method has a peak in 45-60.

Table 3. Comparing output meshes produced by tetGEN, *Delmesher-e4* and *Delmesher* for model 71331. Average min indicates that the absolute minimum is computed on each single triangle and the average of all these minima is reported. Similarly for Average max.

	TetGEN	*Delmesher-e4*	*Delmesher*
Number of vertices	4179	21870	91764
Number of tetrahedra	25255	134259	543356
Min face angle	0.02	0.11	0.17
Max face angle	180	179.7	173.2
Min dihedral angle	0.003	0.008	0.006
Max dihedral angle	180	179.977	179.957
Average min face angle	24	36	33
Average max face angle	104	66	67
Average min dihedral angle	21	33	32
Average max dihedral angle	128	111	111

Limitations. Further research is necessary to make our method faster. Our current prototype uses a particularly cumbersome data structure whose maintenance throughout execution requires time. Furthermore, though we exactly conform with the input, in some cases nearly coplanar acute triangles are chamfered, whereas they would remain unaltered in case of exact coplanarity. Besides making everything slower, this fact causes the creation of more tetrahedra than necessary. This issue was first studied in [Si and Shewchuk 2014] but we aim at full robustness and we argue that it might be obtained through a change of representation as in [Bernstein and Fussell 2009], where nearly-coplanar triangles can be represented by a single plane. Our heuristic approach to handle slivers deserves a deeper understanding. Indeed, it does not prevent convergence in practice, but we still do not fully understand why.

8.3 Conclusion

We have shown that polyhedral surfaces can be effectively chamfered to remove all the acute angles, so that Delaunay refinement is guaranteed to converge. In practice, this idea turned out to be surprisingly effective, and our prototype implementation produces high quality meshes while being more reliable than existing approaches. To the best of our knowledge, this is the first method which is able to produce quality meshes on all the valid models in the Thingi10k dataset.

While the intermediate mesh generated by Delaunay refinement has formal quality guarantees, our final mesh may incorporate low-quality elements near the acute angles in the input PLC. It would be desirable that all the output angles are $\geq min\{\alpha, \delta\}$, where α is the smallest input angle and δ is the target angle. Unfortunately, no existing algorithm (including ours) provides this guarantee, and looking for improvements in this sense is an exciting direction for future research. More practically, we argue that our resulting meshes can be further improved by applying local operators as done, e.g., in [Hu et al. 2018] while preserving the boundary constraints.

We believe our contribution is a crucial step toward meshing algorithms that are provably robust and guarantee the production of high-quality meshes.

ACKNOWLEDGMENTS

This work was partially supported by the MUR-PRIN Project N. 2022YB4NRS "FabDesign", the PNRR Project "RAISE (Robotics and AI for Socio-economic Empowerment)", the NSF grant OAC-2411349, and the NYU IT High Performance Computing resources, services, and staff expertise.

REFERENCES

P. Alliez, C. Jamin, L. Rineau, S. Tayeb, J. Tournois, and M. Yvinec. 2009. CGAL 6.0.1 - 3D Mesh Generation. *CGAL User Manual* (2009). https://doc.cgal.org/latest/Mesh_3/index.html

M. Attene. 2020a. Indirect Predicates. *github* (2020). https://github.com/MarcoAttene/Indirect_Predicates

Marco Attene. 2020b. Indirect predicates for geometric constructions. *Computer-Aided Design* 126 (2020), 102856.

Marco Attene, Daniela Giorgi, Massimo Ferri, and Bianca Falcidieno. 2009. On converting sets of tetrahedra to combinatorial and PL manifolds. *Computer Aided Geometric Design* 26, 8 (2009), 850–864.

M. Bern, D. Eppstein, and J. Gilbert. 1990. Provably good mesh generation. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 231–241 vol.1. https://doi.org/10.1109/FSCS.1990.89542

Gilbert Bernstein and Don Fussell. 2009. Fast, exact, linear booleans. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1269–1278.

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.

Jonathan R. Bronson, Joshua A. Levine, and Ross T. Whitaker. 2013. Lattice Cleaving: Conforming Tetrahedral Meshes of Multimaterial Domains With Bounded Quality. In *Proceedings of the 21st International Meshing Roundtable*. Springer Berlin Heidelberg, Berlin, Heidelberg, 191–209. https://doi.org/10.1007/978-3-642-33573-0_12

Siu-Wing Cheng, Tamal K Dey, Edgar A Ramos, and Tathagata Ray. 2004. Quality meshing for polyhedra with small angles. In *Proceedings of the twentieth annual symposium on Computational geometry*. 290–299.

Siu-Wing Cheng and Sheung-Hung Poon. 2003. Graded conforming Delaunay tetrahedralization with bounded radius-edge ratio. In *SODA*, Vol. 3. 295–304.

L Paul Chew. 1987. Constrained delaunay triangulations. In *Proceedings of the third annual symposium on Computational geometry*. 215–222.

L. Paul Chew. 1993. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry* (San Diego, California, USA) *(SCG '93)*. Association for Computing Machinery, New York, NY, USA, 274–280. https://doi.org/10.1145/160985.161150

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*. 199–208.

Lorenzo Diazzi, Daniele Panozzo, Amir Vaxman, and Marco Attene. 2023. Constrained Delaunay Tetrahedrization: A Robust and Practical Approach. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–15.

Crawford Doran, Athena Chang, and Robert Bridson. 2013. Isosurface Stuffing Improved: Acute Lattices and Feature Matching. In *ACM SIGGRAPH 2013 Talks on - SIGGRAPH '13*. ACM Press, New York, NY, USA, 38:1–38:1. https://doi.org/10.1145/2504459.2504507

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.* (2019).

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 (Aug. 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.

Benoit Hudson, Gary Miller, and Todd Phillips. 2006. Sparse voronoi refinement. In *Proceedings of the 15th International Meshing Roundtable*. Springer, 339–356.

François Labelle and Jonathan Richard Shewchuk. 2007. Isosurface Stuffing: Fast Tetrahedral Meshes With Good Dihedral Angles. In *ACM SIGGRAPH 2007 papers on - SIGGRAPH '07*. ACM Press, New York, NY, USA, 57. https://doi.org/10.1145/1275808.1276448

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

G.L. Miller, S.E. Pav, and N.J. Walkington. 2005. When and why Ruppert's algorithm works. *International Journal of Computational Geometry and Applications* 15 (2005), 25–54. Issue 1.

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 E Pav and Noel J Walkington. 2005. Delaunay refinement by corner lopping. In *Proceedings of the 14th International Meshing Roundtable*. Springer, 165–181.

Alexander Rand and Noel Walkington. 2009. Collars and intestines: Practical conforming Delaunay refinement. In *Proceedings of the 18th International Meshing Roundtable*. Springer, 481–497.

J. Ruppert. 1995. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms* 18, 3 (1995), 548–585. https://doi.org/10.1006/jagm.1995.1021

Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. 2018. Decoupling simulation accuracy from mesh quality. *ACM transactions on graphics* (2018).

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* (Los Angeles, California). ACM Press, 896–904.

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

Jonathan Richard Shewchuk. 1998. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry* (Minneapolis, Minnesota, USA) *(SCG '98)*. Association for Computing Machinery, New York, NY, USA, 86–95. https://doi.org/10.1145/276884.276894

Jonathan Richard Shewchuk. 2000. Mesh generation for domains with small angles. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry* (Clear Water Bay, Kowloon, Hong Kong) *(SCG '00)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/336154.336163

Jonathan Richard Shewchuk. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry* 22, 1-3 (2002), 21–74.

Jonathan Richard Shewchuk. 2008. General-dimensional constrained Delaunay and constrained regular triangulations, I: Combinatorial properties. In *Twentieth Anniversary Volume: Discrete & Computational Geometry*. Springer, 1–58.

J. R. Shewchuk. 2013. Lecture notes on Geometric Robustness. *L.N. of University of California at Berkeley* (2013).

Jonathan Richard Shewchuk and Hang Si. 2014. Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement. In *Proceedings of the thirtieth annual symposium on Computational geometry*. 290–299.

Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw* 41, 2 (2015), 11.

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

B Wördenweber. 1984. Finite element mesh generation. *Computer-Aided Design* 16, 5 (1984), 285–291.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

A FILTERED PREDICATES TO CLASSIFY ACUTE ANGLES

In the remainder, $\mathbf{p}_x$, $\mathbf{p}_y$ and $\mathbf{p}_z$ denote the Cartesian coordinates of a point $\mathbf{p}$.

A.1 dotProductSign3D

Our predicate `dotProductSign3D(p,q,r)` robustly calculates the sign of the dot product $d = (\mathbf{p} - \mathbf{q})(\mathbf{r} - \mathbf{q})$. Calculations are done using FP arithmetic and the sign of d is guaranteed correct if $|d| > \epsilon_d$, where:

$$\begin{aligned} \mathbf{p'} &= \mathbf{p} - \mathbf{q} \\ \mathbf{r'} &= \mathbf{r} - \mathbf{q} \\ \delta_d &= max\{|\mathbf{p'}_x|, |\mathbf{p'}_y|, |\mathbf{p'}_z|, |\mathbf{r'}_x|, |\mathbf{r'}_y|, |\mathbf{r'}_z|\} \\ \epsilon_d &= 1.443289932012704 * 10^{-15} \delta_d^2 \end{aligned}$$

If $|d| \leq \epsilon_d$ we switch to more precise number types and eventually to exact arithmetic [Attene 2020b].

A.2 dihedralSign

Our predicate `dihedralSign(a,b,c,d)` robustly calculates the sign of:

$$h = [(\mathbf{b} - \mathbf{d}) \times (\mathbf{c} - \mathbf{d})][(\mathbf{a} - \mathbf{d}) \times (\mathbf{c} - \mathbf{d})]$$

Even in this case we use FP arithmetic and the sign of h is guaranteed correct if $|h| > \epsilon_h$, where:

$$\begin{aligned} \mathbf{a'} &= \mathbf{a} - \mathbf{d} \\ \mathbf{b'} &= \mathbf{b} - \mathbf{d} \\ \mathbf{c'} &= \mathbf{c} - \mathbf{d} \\ \delta_h &= max\{|\mathbf{a'}_x|, |\mathbf{a'}_y|, |\mathbf{a'}_z|, |\mathbf{b'}_x|, |\mathbf{b'}_y|, |\mathbf{b'}_z|, |\mathbf{c'}_x|, |\mathbf{c'}_y|, |\mathbf{c'}_z|\} \\ \epsilon_h &= 1.332267629550189 * 10^{-14} \delta_h^4 \end{aligned}$$

All the filter values were derived using the JPCK tool in [Attene 2020a].

B ORTHOGONAL CHAMFERING THEOREM

We prove that the surface produced by the chamfering (Sect. 4.2) has no acute angles. The core of the proof is represented by the following two statements.

Statement 1. All the *bridge* vertices introduced during *vertex chamfering* have no incident acute angles.

proof. Notation refers to Fig. 14 left. Consider a triangle t whose vertex V was chamfered, let α be the amplitude of the angle at V. By construction:

- both internal angles at A and both internal angles at B are right angles,
- segments VA and VB have the same length,
- segments AP and QB have the same length.

Being $VAPQB$ a pentagon, the sum of its internal angles is 540°. Thus, the (red) angles at P and Q facing V are congruent and each of amplitude $180° - \alpha/2$. Also, the (blue) angles at P and Q not-facing V are congruent too, and each of amplitude $180° + \alpha/2$. Being $0 < \alpha < 180°$ the statement follows. □

Statement 2. The endpoints of an edge created during *edge chamfering* have no incident acute angles.

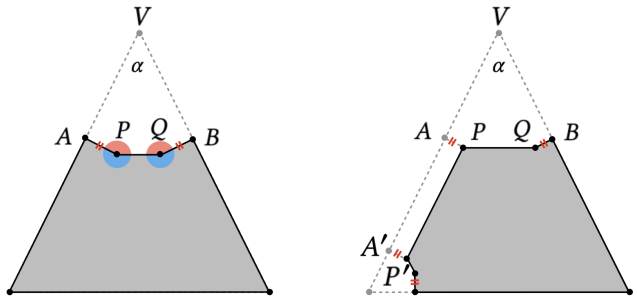


Fig. 14

proof. Notation refers to Fig. 14 right. As long as segments AP and $A'P'$ have the same length and both are orthogonal to AA' the quadrilateral $PAA'P'$ is a rectangle. Consider P, one of the endpoints of the new edge PP'. The angle $\widehat{APQ}$ is not modified by *edge chamfering*, while angle $\widehat{QPP'}$ is reduced by 90°. Recalling the proof of the previous statement, it is possible to conclude that the amplitude of $\widehat{QPP'}$ is $90° + \alpha/2$. A symmetrical argument can be applied at P' and the statement follows. □

These two statements ensure that no acute angles are created while chamfering each input triangle. To conclude, let us verify that no acute angles are created between any two adjacent triangles. Consider the case of two triangles t_1 and t_2 sharing an edge UV such that U or V or both of them were chamfered. There exists at least a point A on UV such that a *bridge*-edge AP_1 of t_1 meets a *bridge*-edge AP_2 of t_2. Being both AP_1 and AP_2 orthogonal to UV at A, the angle $\widehat{P_1AP_2}$ will be acute if and only if the dihedral angle at UV between triangles t_1 and t_2 is acute, by the way in such a case *edge chamfering* would remove the acute dihedral angle.

C CASCADED IMPLICIT POINTS

A recent release of the Indirect Predicates library [Attene 2020a] supports implicit points built on top of other implicit points (this was not possible with the old version used in the original CDT code). Our changes to integrate this new feature in the CDT code amount to just removing a type cast in a few code lines. Indeed, the original CDT expected each LNC to be a combination of two explicit points P_1, P_2, and used these explicit points to create new LNCs. With the new library, P_1 and P_2 need no longer be explicit and we removed the type cast which was forcing that.