Dr. Delaunay or: How I Learned to Stop Worrying and Love the Triangulations

Category: Research

Figure 1: To be updated.

Abstract

We attempt a comprehensive and visual summary of geometry processing, surface, and volumetric meshing. We investigate methods such as Delaunay triangulations, Delaunay tetrahedrizations, and intrinsic triangulations. We also review the impact of geometry processing (e.g., surface simplification, mesh refinement, etc.) on the resulting triangulated or tetrahedrized meshes.

Index Terms: Mesh Generation—Volume Meshing—Surface Meshing—Geometry Processing;

1 Introduction

Triangle and tetrahedral meshes are widely used in the field of computer graphics, architecture, and engineering. In most cases, the complexity of mesh generation is hidden by the end-users. Meshes can be obtained from 3D reconstruction[1], modeling or can be simply found in the *wild*. However, unless it is modeled by a professional, the quality aspect is often questionable. Moreover, if the input consists solely of points in space, then we often need sophisticated algorithms for generating high-quality meshes.

In 2D, the Delaunay triangulation (DT) (Sec. 2.1) has been extremely effective in obtaining high-quality and robust meshes. The use of DT has also been extremely helpful for problems that require solving partial differential equations (PDEs). However, in 3D, although there has been immense work accomplished, robust triangulation or tetrahedrization is still an active area of research. The 3D tetrahedralization of smooth implicit surfaces is quite mature (Hu et al., 2018). Nonetheless, existing techniques are often not constrained to input meshes, and if so, the expected inputs are often simple meshes.

Recently (Hu et al., 2018) proposed a robust tetrahedral meshing approach that required no manual intervention. Similarly, (Diazzi et al., 2023) introduced robust calculation of constrained Delaunay Tetrahedrizations of a piecewise-linear complex. [ID0: To be extended.] In this report, we review these works extensively. In summary, our contributions are:

[1]In a concurrent report we discuss the practical aspects of 3D reconstruction. [ID0: to be updated with the 3D vision report]

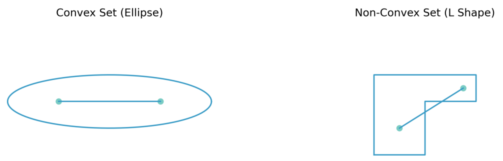


Figure 2: Example of a convex (left) and non-convex (right) set.

- A detailed and visual introduction to Voronoi and Delaunay Diagrams.
- Review of the current state of the art in (constrained) triangle and tetrahedral meshing.
- A brief overview of modern geometry processing techniques.

Concerning the academic tone and language. We rebel against the notion that a scientific work must be written with a certain authority. We embrace our ineptitude in some of the topics that we discuss. To best our knowledge and ability, we will try to summarize the current state of the art.

2 Background

We begin by providing some definitions that will be helpful for the rest of this report. To clarify some of the abstract notions, we will also attempt to provide informal explanations alongside visualizations.

2.1 Voronoi and Delaunay Diagrams in 2D

In this section, we introduce Voronoi diagrams and Delaunay triangulations for 2D. Our discussion will mainly follow the work of (Edelsbrunner, 2013; Edelsbrunner et al., 2006).

Convex polygons. Given two points x, y from the set X. Provided that every point on the line segment that connects x, y belongs to X, then X is convex (Fig. 2).

Lemma 2.1. *The intersection of convex sets is convex.*

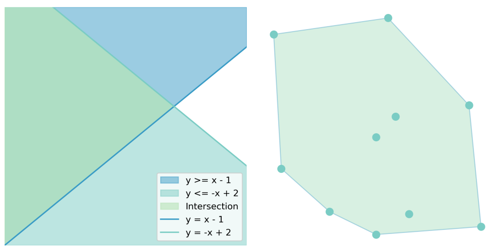


Figure 3: Intersection of two Half-Planes (left) and convex hull (right) of a point set.

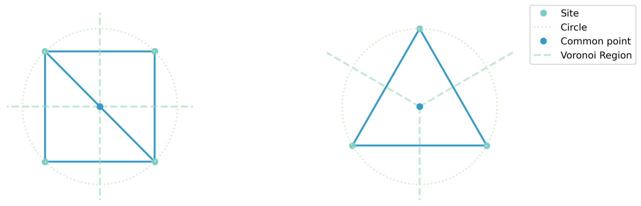


Figure 5: Left: Four Voronoi edges meet at a common point (i.e., degeneracy). Right: Three Voronoi edges meet at a common point (i.e., general case).

Every point along the line segment belongs to the intersection, if points x and y belong to the intersection. A half-plane is a simple convex set. It contains all the points on (one side of) the line.

A *convex polygon* is the intersection of multiple half-planes (Lemma 2.1). The *convex hull* is the intersection of all convex sets containing vertices. Intuitively, one can think of a rubber band that tightly snaps a collection of points (Fig. 3).

Voronoi Diagrams. Given a collection of points from a set S. To distinguish them from other points in the plane, we name these *sites*. Each site s has a *Voronoi region* V_s. A Voronoi region of a site describes the region of points that are closer to that site than to any other site. More concretely, V_s is defined as

$$V_s = \{x \in \mathbb{R}^n \mid ||x-s|| \leq ||x-t||, \forall t \in S\}. \tag{1}$$

Given a set of points satisfy $||x-s|| \leq ||x-t||$, then it is a closed half-plane. Hence, V_s is the intersection of many-half planes and a convex polygon. The *Voronoi diagram* of set S is defined as the set of Voronoi regions (Fig. 4).

Delaunay triangulations. The Delaunay triangulation (DT) is the dual graph of the Voronoi diagram. Given the Voronoi diagram of $S \subset \mathbb{R}^2$, and assuming two Voronoi regions share an edge, we connect the two sites by a straight edge (Fig. 4). From three sites, we get a triangle in the DT. Given no four sites are in a common circle, then sites in S are in *general position*. We often assume this is the case, but if not, we call this a *special case*, or *degeneracy*.

Degeneracy. Degeneracy arises if four or more Voronoi regions share a common point u (Fig. 5). The points generating the four—or, more—regions have all the same distance to u (i.e., they all lie on a circle in which the center is u). An arbitrary small perturbation can be applied to remove the degeneracy (Edelsbrunner, 2013).

Circumcircle and Delaunay Triangle. Assuming general position, the circumcircle passes all vertices of a Delaunay triangle abc. The center of the circumcircle is the point where three Voronoi regions intersect (Fig. 5; right), i.e., $u = V_a \bigcap V_b \bigcap V_c$. The radius of the circle is $r = ||u-a|| = ||u-b|| = ||u-c||$. If the circle does not contain any point of S, we call the circle empty. Empty circles characterize Delaunay triangles. More formally, let $S \subset \mathbb{R}^2$ and in general position, and let $a,b,c \in S$ be three sites. Then, abc is a Delaunay triangle if and only if (*iff*) the circumcircle is empty.

Supporting Circle. An edge owned by one (or two) Delaunay triangle(s) is called a Delaunay edge. Just as an empty circle passes through the vertices of a Delaunay triangle, an empty circle passes through the endpoints of a Delaunay edge (Fig. 6). More formally, let $S \subset \mathbb{R}^2$, in general position, and $a,b \in S$. Then ab is a Delaunay edge *iff* an empty circle passes through a and b.

Local Delaunay. The points in S are triangulated K, if the triangles subdivide the convex hull of S. (1) Given an edge $ab \in K$ owned by a single triangle (i.e., bounded by its convex hull), or, (2) shared by two triangles, abc and abd, where d is not in the circumcircle of abc, then the edge is *locally Delaunay* (Fig. 8). Using the local Delaunay property, we introduce the *Delaunay Lemma*:

Lemma 2.2. *Given a triangulation K of a set S, if every edge is locally Delaunay, then K is the DT.*

Edge flip. The union of two triangles abc and abd is a convex quadrangle. If the shared edge ab is non-locally Delaunay, we flip ab to cd (Fig. 8). Given a non-DT, we can convert it to Delaunay, by *flipping* non-locally Delaunay edges to local Delaunay edges. More informally, an edge satisfies the Delaunay property *iff*, $\theta_a + \theta_b \leq \pi$, where θ_a is the opposite angle of a shared edge, and θ_b is the opposite angle on the corresponding face (Sharp et al., 2021). The edge-flip algorithm can be found in many literature (e.g., (Edelsbrunner et al., 2006)).

Maximizing the minimum angle. By an edge flip, we replace two old triangles with two new ones. Consequently, we also replace six angles with six new ones. One of the old angles must be at least

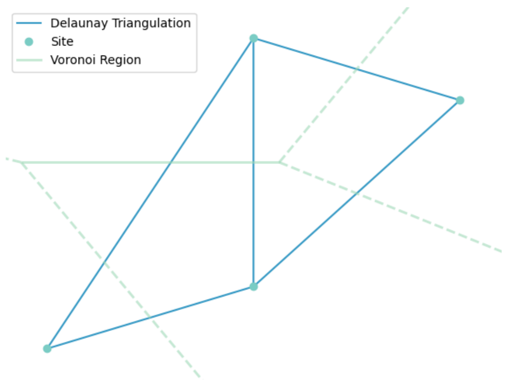


Figure 4: Voronoi diagram and the dual Delaunay triangulation.

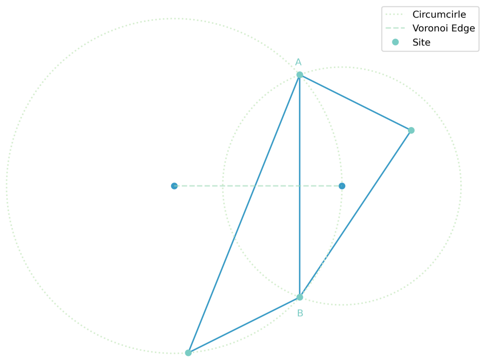


Figure 6: A Delaunay edge ab, and supporting circles.

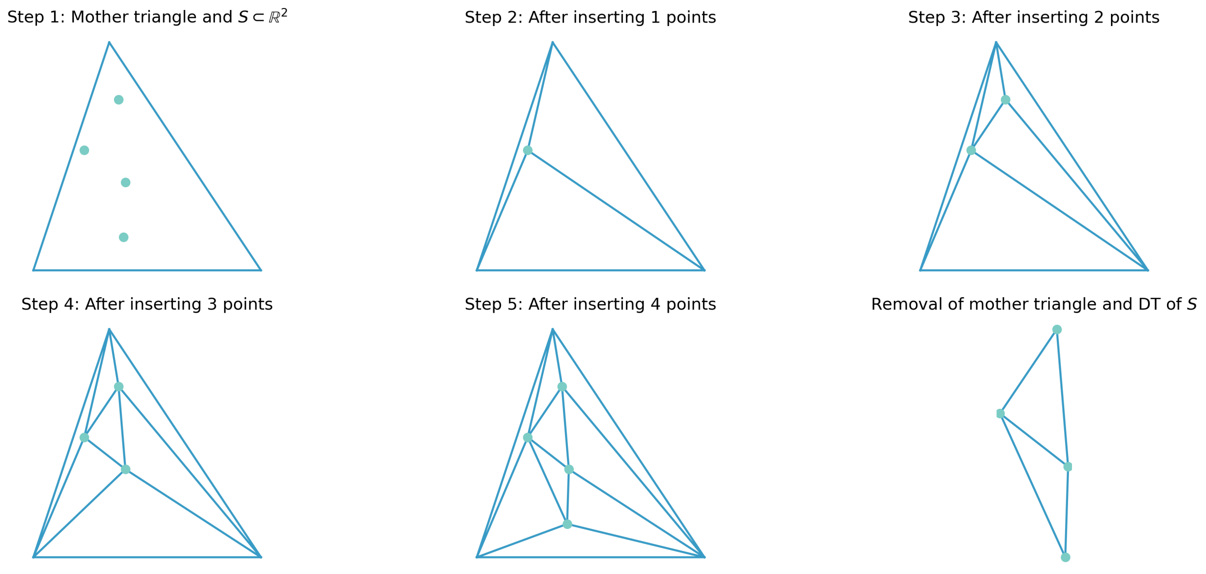

Step 1: Mother triangle and $S \subset \mathbb{R}^2$
Step 2: After inserting 1 points
Step 3: After inserting 2 points
Step 4: After inserting 3 points
Step 5: After inserting 4 points
Removal of mother triangle and DT of S

Figure 7: Delaunay triangulation of $S \subset \mathbb{R}^2$. In this example, S consists of 4 sites.

as small as one of the new ones. Therefore a flip does not decrease the smallest angle in a non-DT. This means the smallest angle in non-DT is no larger after converting it into a DT. A DT maximizes the minimum angle among all triangulations of a point set $S \subset \mathbb{R}^2$.

Incremental DT construction. We briefly introduce an incremental algorithm for constructing DT using randomization and edge flips. Let sites in the set $S \subset \mathbb{R}^2$ be $p_1, p_2, ..., p_n$. For any collection of n sites, instead of inserting them sequentially, we shuffle the order of their insertions. This leads to $O(n)$ expected total number of structural changes to the diagram, and $O(n \log n)$ overall cost of the algorithm (Guibas et al., 1992). We know that if we add a new site to S it can be either inside or outside the convex hull of S (Fig. 2). We reduce the two possibilities to only an inside case. This is done by defining a *mother triangle xyz*, which initializes the triangulation D_0. The mother triangle wraps the sites of S. We define $S_i = \{x, y, z, p_1, p_2, ..., p_i\}$, and let D_i the triangulation of S_i. The algorithm for incremental construction is defined in (Alg. 1). An example of DT construction can be found in (Fig. 7).

Algorithm 1 Incremental DT construction (Guibas et al., 1992)

```
for $i \leftarrow 1$ to $n$ do
    find $\tau_{i-1} \in D_{i-1}$ containing $p_i$
    add $p_i$ by splitting $\tau_{i-1}$ into three
    while $\exists ab$ not locally Delaunay do
        flip $ab$ to other diagonal $cd$
    end while
end for
```

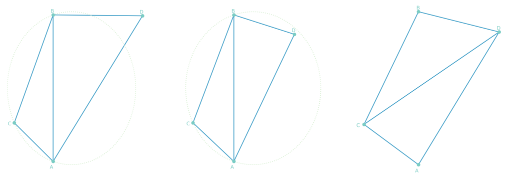


Figure 8: Left: ab is locally Delaunay. Middle: ab is not locally Delaunay. Right: Result of an edge flip makes ab locally Delaunay.

Constrained Delaunay Triangulations. In constrained triangulations, the input not only consists of a set of sites $S \subset \mathbb{R}^2$, but also a set of line segments L (Fig. 9). The input is triangulated as before, but we ensure all line segments are contained as edges in the triangulation. A constrained triangulation can be constructed by connecting the sites of S by a straight edge, *iff* the edge does not intersect a line segment from L. A constrained DT is not truly a DT. Not all resulting triangles are Delaunay triangles. But, it avoids small angles as much as possible. In some sense, it is the closest to a DT.

With constrained DT, we further generalize the concept of locally Delaunay and introduce (Lemma 2.3). Given a constrained triangulation K of S and L. An edge $ab \in K$ is *locally Delaunay*, (1) if $ab \in L$, or (2) ab is a convex hull edge, (3) or d lies outside the circumcircle of abc, where abc, and $abd \in K$.

Lemma 2.3. *Given every edge of K is locally Delaunay, then K is the constrained Delaunay triangulation of S and L.*

Constrained DT can be constructed just as before, but the edges from L do not need to be flipped due to the generalized definition above. As before, constrained DT maximizes the minimum angle.

Conforming Delaunay Triangulations. Similar to constrained DT, the input of a conforming DT are sites S, and line segments L. However, a conforming DT is truly a DT (i.e., all triangles are Delaunay). To truly become DT, a conforming DT may subdivide its edges (Fig. 9). This subdivision is done by inserting new points, called *Steiner points*. Consequently, Steiner points are vital for maintaining the Delaunay property. But, they are also used to meet constraints such as minimum angle and maximum triangle area.

Constrained Conforming Delaunay Triangulations. Constrained conforming DT is a hybrid approach of constrained and conforming DTs. They also use Steiner points for the triangulation (Fig. 9). However, unlike conformal DTs, they are not truly Delaunay. For this reason, usually constrained conforming DTs use fewer Steiner points than conforming DTs, which makes them more preferable in practice.

Having discussed the 2D case, we could extend our discussion to the n-dimensional case. However, we remain hesitant and will only concentrate on the 3D case. As we will see shortly, in the 3D case, the concepts we introduced here not only gets more difficult to visualize but will also start to become more susceptible to failure cases.

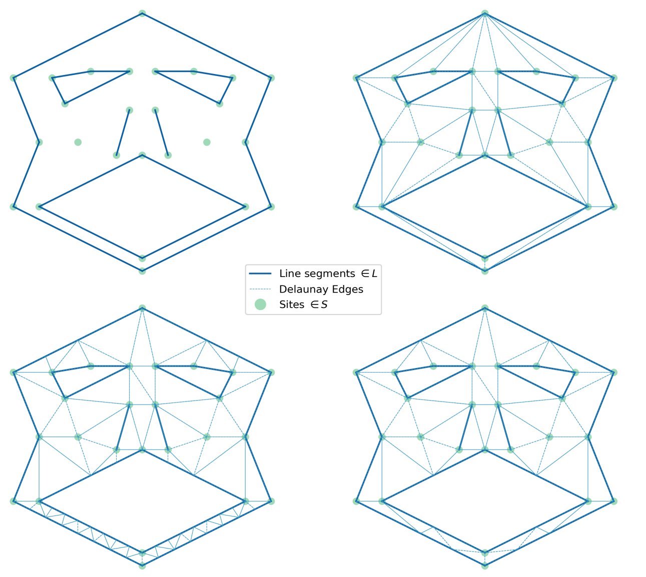


Figure 9: Input (top left), constrained DT (top right), conforming DT (bottom left), and constrained conforming DT (bottom right). We used the *Triangle* library (J. R. Shewchuk, 1996) for visualization.

2.2 Voronoi and Delaunay Diagrams in 3D

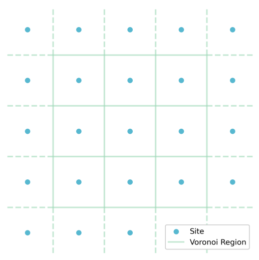
We now move on to the 3D case. Delaunay triangulations are broadly used as a generic term even for higher dimensions. For the sake of clarity, we refer to the 3D case as Delaunay Tetrahedrizations (DTET).

Another distinction to be made is the projection of points in 3D to 2D. For example, given the task is to reconstruct a surface using the points in 3D—rather than a volumetric representation. Then, one could project the 3D points onto a 2D plane and perform Delaunay triangulations. However, in that case, the task essentially becomes a 2D Delaunay triangulation, and the main difficulty is to find an accurate representation of 3D points in 2D. For now, we solely focus on DTETs.

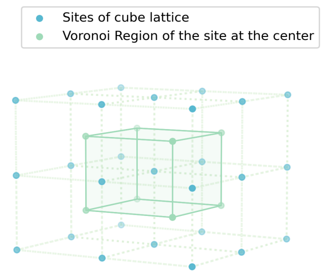
Lattices. Given set of d linearly independent vectors in $\mathbb{R}^d$, we generate a lattice made up by all integer combinations $S = \{\sum_{i=1}^{d} k_i u_i \mid k_i \in \mathbb{Z} \forall i\}$. In 2D, the lattice is defined using two vectors. For example, $u_1 = (1,0)$ and $u_2 = (1,0)$ generates the *square lattice* (Fig. 10a). Given two sites $a, b \in S$, if we translate the lattice by the difference vector we get $S = S + (b - a)$, which leaves the lattice invariant. Similarly, the Voronoi regions remain the same $V_b = V_a + (b - a)$. This implies that the Voronoi region of the origin is symmetric.

Cube lattice. The cube lattice is a set of all integer combinations of three vectors: $(2,0,0), (0,2,0), (0,0,2)$. As discussed previously, in 2D no four sites should lie in a common circle for sites to be in general position. The Voronoi diagram in $\mathbb{R}^3$ requires that no five points should lie on a common sphere. Both sites in square and cube lattice are not in general position. Consequently, if we dualize the Voronoi diagram we do not get a triangulation. Rather, we get cubes that tile $\mathbb{R}^3$ (Fig. 10b). Note that, as previously, the vertices of the cube are the vertices of the DTET and centers of Voronoi regions.

Body-centered cube lattice. By adding the centers of the cube to the set, we get a body-centered cube lattice (BCC). We generate a BCC from three vectors $u_1 = (1,1,-1)$, $u_2 = (1,-1,1)$, and $u_3 = (-1,1,1)$. The relationship to the cube lattice can be shown by the pairwise sums: $u_1 + u_2 = (2,0,0)$, $u_1 + u_3 = (0,2,0)$ and $u_2 + u_3 = (0,0,2)$. The Voronoi region of the origin of a BCC lattice wraps all points closer to the origin than any other point. It is drawn by



(a) Voronoi diagram of square lattice.



(b) A subset of sites in the cube lattice, and the Voronoi region of the site at the center.

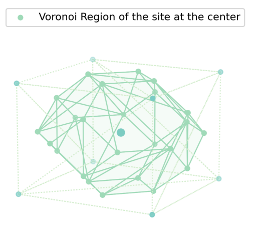
Figure 10: Lattices and Voronoi regions.

intersecting a cube with an octahedron. The resulting shape is a truncated octahedron (Fig. 11a). Dualizing the Voronoi diagram, we get the DTET of the BCC lattice. The resulting triangulation is in fact a tetrahedron. The long edges of the tetrahedron are constructed by connecting the centers of the two neighboring cubes (Fig. 11b). Similarly, the shorter edges are obtained by connecting the center of the cubes to one of its corners. After performing DTET, every tetrahedron we obtain consists of two long and four short edges.

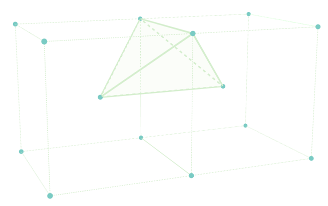
Voronoi Diagrams. Formally, the Voronoi region of a site in $S \subseteq R^3$ is defined as (Eq. 1). In the 3D case, the Voronoi region V_s is the intersection of closed *half-spaces*. Alternatively, V_s is a *convex polyhedron* just like the one on (Fig. 11a). Voronoi regions contains shared facets, edges, and vertices. As before, the set of Voronoi region(s) makes up the Voronoi diagram. Assuming general position, if we have k Voronoi regions, and k points that belong to each region, then k points lie in a common sphere. Since we assume general position, then $k \leq 4$. In other words, a *common point* (Fig. 5) belongs to at least four Voronoi regions.

Delaunay Tetrahedrization. As in the 2D case, the DTET is the dual operation of the Voronoi Diagram. The sites of the Voronoi regions correspond to the vertices of the DTET. The edges of the DTET connects sites of Voronoi regions that share a common facet. The facets of the DTET connects Voronoi regions shared by a common edge. If sites are in general position, then (i) each edge of DTET is shared by three Voronoi regions, (ii) the facets of DTET are triangles, (iii) each vertex of DTET is shared by four Voronoi regions, and (iv) the result of Delaunay consists of tetrahedra

We previously discussed constrained DT for 2D. Throughout the years, the 3D case has been studied extensively. However, unlike the 2D case, it has been a much more challenging problem. We will now continue with a deep dive into tetrahedrizations. First, we will review boundary-approximating tetrahedrizations, and then move on to the constrained DTETs.



(a) Voronoi region of a site in the BCC lattice.



(b) A tetrahedron in the DTET of the BCC lattice.

Figure 11: Voronoi region of BCC lattice and its DTET.

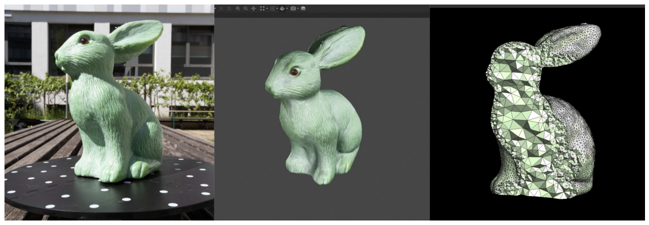


Figure 12: Sapo's bunny found in the *wild* (left), surface reconstruction using Metashape (AgiSoft, 2022) (middle), resulting tetrahedrization (Hu et al., 2018) (right).

3 Approximately Constrained Tetrahedrizations

As the name suggests, boundary-approximating constrained tetrahedrizations aim to construct an approximate tetrahedral mesh from the given input set of triangles (Fig. 12).

Most recently, (Hu et al., 2018) made a giant leap forward in this field. But, previous work such as *Quartet* (Bridson & Crawford, 2014; Labelle & Shewchuk, 2007), or *CGAL* (Fabri & Pion, 2009) is still relevant to this day. Nevertheless, we will mainly follow the work of (Hu et al., 2018): *TetWild*.

Although, there have been improvements in robust triangulation of the interior of a given triangle surface mesh, many existing algorithms either failed or required manual intervention. (Hu et al., 2018) introduced a new approach that made no assumptions on the input mesh (e.g., the existence of self-intersections, watertightness, etc.). Their approach mainly focuses on robust tetrahedrizations that require no manual intervention of the input. Rather than expecting clean inputs, they underlie that meshes found in the wild often resemble a triangle soup. This recognition brings the necessity of viewing mesh repair as an integral part of the constrained tetrahedrization problem. Overall, (Hu et al., 2018) views these key aspects in their approach:

- The provided input will be more likely imperfect. Deviations from the input within a user-defined envelope ε can be tolerated.
- Reformulating the meshing problem according to the input is a necessity.
- Robustness is the primary priority. Given robustness requirement is not broken, quality improvements can be included.
- For practical purposes, the method must output in floating-point coordinates. However, for robustness, in-between steps may use exact rational computations even though it might influence the overall performance.

Having introduced (Hu et al., 2018)'s main goal and perspective, we will now discuss their methodology.

3.1 Constructing Approximately Constrained Tetrahedrizations

We continue with a deep dive into the construction of approximately constrained tetrahedrizations. An intuitive summary of (Hu et al., 2018) approach for 2D is shown in (Fig. 15).

Problem statement. The input is assumed to be an arbitrary triangle soup. The user-specified parameters are envelope tolerance ε ($b \times \varepsilon$, where b is the bounding box diagonal) and the desired edge length l. The method aims to construct an *approximately constrained tetrahedrization*. The resulting tetrahedrization of the input triangle mesh must be (1) bounded within the specified ε, (2) the edge lengths must be bounded by desired l and, (3) it must not contain any inverted elements. Given that the first and third requirements are fulfilled, then mesh is defined as *valid*. An example of choosing the default (0.001) and a high (0.1) ε rate is shown in (Fig. 13).

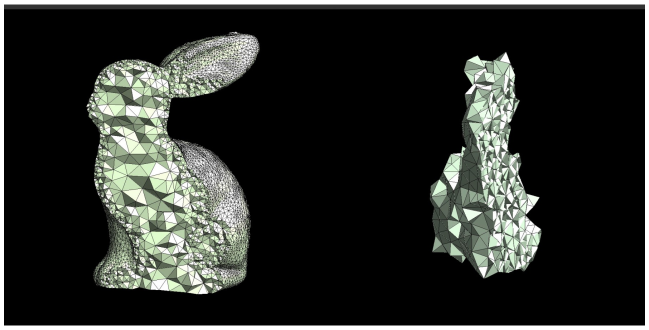


Figure 13: Using the default (0.001) ε value allows us to capture the features of the mesh and increases geometric fidelity (left). Picking an arbitrary high (0.1) value results in failure.

Approach overview. The approach of (Hu et al., 2018) consists of two phases. In the first phase, they generate a valid mesh using exact rationals without hinging on its geometric quality. This phase consists of Delaunay tetrahedrization, a volumetric Binary Space Partitioning (BSP) tree construction, and barycenter tetrahedrization. The result of this phase is a volumetric mesh that also surrounds the provided model (i.e., tetrahedrization fills the entire bounding box of the model). The second phase ensures improved geometric quality and rounding to floating point numbers. This phase entails mesh optimization (e.g., local operations that refine, coarsen, swap, or smooth the elements) and inside-outside segmentation. Unlike existing algorithms, which seek to construct a high-quality mesh directly, (Hu et al., 2018) argue that their two-phase approach is the primary reason for their success in terms of robustness.

The first phase. We now continue with the intricacies of the first phase. Unlike constrained Delaunay tetrahedrizations (Sec. 4), boundary-approximating—or, in this case, unconstrained— Delaunay tetrahedrizations can be robustly implemented using exact rational numbers (Jamin et al., 2015). Thereby, (Hu et al., 2018) begins the first phase by (1) creating a non-conforming tetrahedral mesh M from the vertices of the triangle soup using exact rational kernels in CGAL (Jamin et al., 2015). Next, to ensure conformity, (Hu et al., 2018) follows a similar approach to (Joshi & Ourselin, 2003). They assume that each tetrahedron is the root of BSP. (2) This allows them to intersect the triangles from the input geometry with all the tetrahedra from M. More informally, each triangle from the input is assumed to be a plane. And, they intersect these planes with all the tetrahedra in M. Since this operation is done under exact rationals, robustness is ensured even for a degenerate input. The result of this operation is a polyhedral mesh. (3) To convert it into a tetrahedral one, they first triangulate all the faces. Then, at the barycenter, they add a vertex and connect it to all the triangular faces. Given at least four vertices are linearly independent, then all

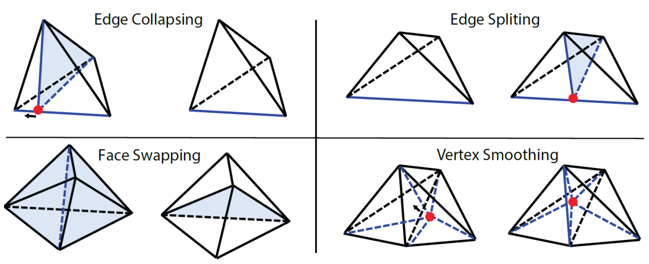


Figure 14: (Hu et al., 2018) employs four local operators for mesh improvement. These are: edge collapsing, edge splitting, face swapping and vertex smoothing.

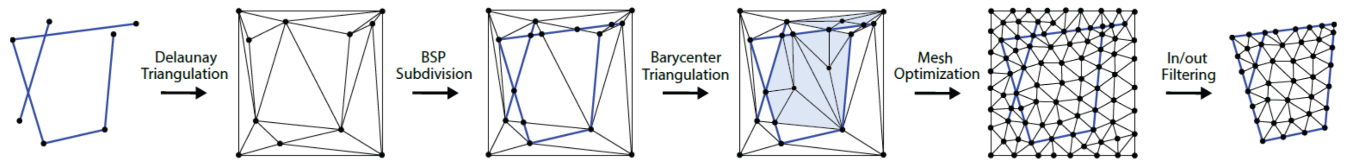


Figure 15: The approach of (Hu et al., 2018) consists of two phases. In the first phase, they generate a valid mesh using exact rationals without hinging on its geometric quality. This phase consists of Delaunay triangulation, BSP-tree construction, and barycenter triangulation. The second phase ensures improved geometric quality and rounding to floating point numbers. This phase entails mesh optimization and filtering out the elements outside of the domain. Figure is taken from (Hu et al., 2018) and represents the 2D case for simplicity.

convex cells are non-degenerate (i.e., tetrahedra connected to the barycenter will be non-degenerate). The result of this phase is *valid* a tetrahedral mesh that conforms to the input triangles. Although the resulting mesh is valid, it is not guaranteed to be high quality. Hence, the result needs to be improved, which brings us to the second phase.

The second phase. The second phase involves different steps and many details. For the sake of conciseness, we summarize the most relevant details for our discussion. We highly recommend and refer the reader to (Hu et al., 2018) for the pristine discussion. After obtaining a valid, but perhaps poor-quality mesh, it needs improvement. Only after performing the optimization, and ensuring that the validity is preserved the interior of the volume can be extracted. (Hu et al., 2018) employs a greedy optimization pipeline using local mesh improvement operations (Dunyach et al., 2013; Faraj et al., 2016; Freitag & Ollivier-Gooch, 1997). Throughout the optimization, (Hu et al., 2018) avoids operations that introduce inverted tetrahedra whose orientation is negative (also denoted as *Validity Invariant 1*). This is achieved using exact predicates (Brönnimann et al., 2017) for both rational and floating point coordinates. Similarly, the *embedded surface*—which consists of faces of the tetrahedra from the first phase— is tracked. (Hu et al., 2018) ensures that no improvement presents faces that exceed the user-specified envelope ε (*Validity Invariant 2*). More informally, the envelope can be thought of as a blanket with thickness ε that surrounds the embedded surface. It is made sure that the embedded surface is always protected by this *magical* blanket that may violate the *Validity Invariant 2*. To track the quality of the optimization, (Hu et al., 2018) uses the 3D conformal energy (Rabinovich et al., 2017). The 3D conformal energy is especially useful since it is indifferent to isotropic scaling. Moreover, it penalizes flat and fat elements, slivers (Sec. 4), and inversions. (Hu et al., 2018) employs four local operations for mesh improvement: edge splitting and collapsing, face swapping, and vertex smoothing (Fig. 14). They introduce an asymmetric optimization scheme. The mesh is optimized in 4 passes: (1) splitting (refining), (2) collapsing (coarsening), (3) swapping, and (4) smoothing. The local coarsening and optimization operations are only performed if the mesh quality is improved. Similarly, the refinement operator is performed until the user-specified edge-length l is reached, or a region is locked. The optimization finishes if either the maximum energy is sufficiently small (default: less than 10), or the maximum number of iterations is reached (default: 80). (Hu et al., 2018) note that this strategy not only avoids over-refinement but also averts introducing unnecessary vertices. This approach results in high-quality meshes even for poor-quality inputs (Fig. 16). The last part of this phase entails input-output segmentation. (Hu et al., 2018) follows the technique proposed in (Jacobson et al., 2013), which addresses potential imperfections in the embedded surface by an inside-outside function. The defined function is used to extract the interior of the embedded mesh. An advantage of this approach is even if the input mesh contains small gaps or large surface holes, they are filled according to the induced winding number field (Jacobson et al., 2013) (Fig. 17). However, inevitably, if the input mesh has holes, then the resulting tetrahedral mesh will not be bound to the specified ε envelope due to triangles that close the hole. Having discussed the details, we will now make some concluding remarks regarding the work of (Hu et al., 2018).

3.2 Final Remarks on TetWild

We see no justice in repeating the results or in-depth discussion of (Hu et al., 2018) since it deserves a separate read on its own. Their paper, among plentiful visualizations—24 in total—provides a thorough analysis and comparison to previous work. The two-phase approach is indeed set to be *a very useful and revolutionary way*[2] to obtain tetrahedrization of a provided triangle mesh. Using the Thingi10K (Zhou & Jacobson, 2016) dataset, the two-phase approach of (Hu et al., 2018) can produce 9997 valid models (out of 10000) after the first phase and the remaining 3 models after the second phase. However, as this author's hairline, nothing is perfect in life. (Hu et al., 2018) mention that their approach struggles with preserving sharp features. Similarly, although they discuss the possibility of using their approach for repairing and improving meshes (Fig. 16), they note that it is limited to closed surface faces. Finally, they also mention the performance drawbacks to competing methods, though this already seems to be addressed in their next work (Hu et al., 2020).

[2]Spoken during the QA Session after (Hu et al., 2018) presentation at SIGGRAPH.

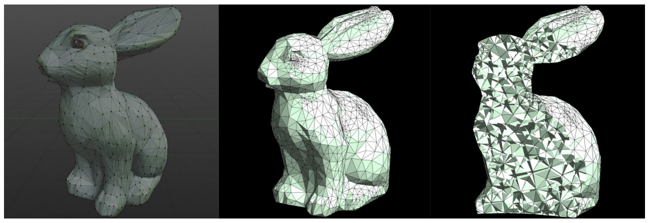


Figure 16: Heavily decimated bunny (left), and its tetrahedrization (Hu et al., 2018) (middle and right).

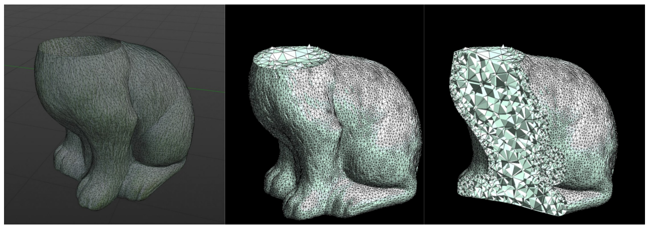


Figure 17: A poor half-eaten bunny (left) and its tetrahedrization (Hu et al., 2018) (middle and right). Even if the input mesh contains holes, it will be closed accordingly.

Although boundary-approximating tetrahedrizations proved extremely useful, they rely on lossy and potentially not bijective projections to map boundary conditions. This makes them less suitable for solving PDEs. We now jump into our time-travel machine and pilgrimage to 2023. The recent work of (Diazzi et al., 2023) aims to construct boundary-preserving constrained Delaunay tetrahedrizations, which by design makes them suitable for solving PDEs.

4 Constrained Delaunay Tetrahedrizations

We now move to constrained Delaunay Tetrahedrizations (DTETs) of a piecewise-linear complex (PLC). We will mainly follow the work of (Diazzi et al., 2023).

Given non-intersecting line segments and vertices, a constrained DT in 2D is always possible (Diazzi et al., 2023). However, the 3D case is not always guaranteed. Consequently, often the input is augmented with *Steiner points* (Sec. 2.1) (Murphy et al., 2001; J. Shewchuk, 2002; Si & Gärtner, 2005). The positioning and amount of these Steiner points is still an open problem (Ruppert & Seidel, 1992).

Over the years, there were different methods to tackle the 3D case (George et al., 1991; Guan et al., 2006; Joe, 1991; J. Shewchuk, 2002; Weatherill & Hassan, 1994). (J. Shewchuk, 2002) presented an algorithm that splits all of the edges that neighbors acute vertices. This led to a provably accurate constrained DTETs that protected acute vertices. (Si & Gärtner, 2005) followed the work of (J. Shewchuk, 2002) that still protected the vertices, but did so using fewer Steiner points. To this day, this approach is the state of the art for calculating constrained DTET in 3D. A widely used tetrahedral meshing software *Tetgen* (J. R. Shewchuk & Si, 2014; Si, 2015) employs a floating point implementation of this algorithm. However, it fails to produce a constrained DTET on around 8.5% of the valid models in the *Thingi10k* (Zhou & Jacobson, 2016) dataset.

Although some of the failure cases *partly* stem from the numerical inaccuracies due to the *tetgen*'s floating-point implementation, (Diazzi et al., 2023) add that the failure *mainly* originates from a theoretical issue due to an incorrect assumption in the work of (Si & Gärtner, 2005). The authors validate this using an exact-number implementation (CORE library (Burnikel et al., 1996; Karamcheti et al., 1999)).

Conforming Delaunay Tetrahedrization. Conforming tetrahedrizations (e.g., (Cohen-Steiner et al., 2004; Murphy et al., 2001)) introduce many Steiner points and tetrahedras. Moreover, this approach might result in elements that do not fulfill the Delaunay criterion (Sec. 2.1). Recently, (Alexa, 2020) investigated the possibility of using weights for input vertices with the expectation that the resulting DTET would need fewer or no Steiner points. Even if those weights exist, (Diazzi et al., 2023) conclude that the computational complexity makes them impractical.

Delaunay refinement. Tetrahedras can be improved by introducing new vertices at the centers of circumscribing spheres (e.g., (Jamin et al., 2015; Ruppert, 1995; J. R. Shewchuk, 1998)). These approaches ensure termination but may allow *slivers* in the resulting tetrahedra. Slivers refer to tetrahedras with close to degenerate volume but with appropriate radius-edge-length ratios. There are various approaches to address slivers (Alexa, 2019; Alliez et al., 2005; Du & Wang, 2003; Tournois et al., 2009). (Diazzi et al., 2023) compare their approach against CGAL.

Robust geometric predicates. A key factor of meshing algorithms is numerical robustness. With the standard floating point arithmetic, methods may fail. Numerical robustness can be achieved using exact arithmetic kernels (Fabri & Pion, 2009), but this approach could be too slow. Other methods aim to guarantee that the program flow is correct with the use of rounded outputs (Li et al., 2005; J. Shewchuk, 1997). However, points that are generated by the algorithm may not guarantee correct program flow. To address this, these points can be expressed as a combination of input points (Attene, 2020). Using this solution, (Diazzi & Attene, 2021) exploited the efficiency of floating point hardware and managed to successfully create polyhedral meshes. However, arbitrarily bad shape of cells limits their usage and brings the necessity of a more advanced approach.

In summary, (Diazzi et al., 2023) contribute by providing a novel algorithm for computing the CDT of a valid PLC on HW accelerated floating-point computation without giving up on robustness. This is achieved by:

1. Avoiding the implicit assumption that all local cavities formed to insert PLC can be sufficiently expanded. This leads to a key property of their algorithm without using irrational coordinates for Steiner points.
2. An exact and efficient implementation of their algorithm using indirect floating-point predicates (Attene, 2020).

Having introduced the main contributions of (Diazzi et al., 2023), we will now discuss their approach in more detail. [ID0: Work in progress...]

References

AgiSoft. (2022). *Metashape* (Version 1.8.4) [https://www.agisoft.com/].

Alexa, M. (2019). Harmonic triangulations. *ACM Trans. Graph.*, *38*(4). https://doi.org/10.1145/3306346.3322986

Alexa, M. (2020). Conforming weighted delaunay triangulations. *ACM Trans. Graph.*, *39*(6). https://doi.org/10.1145/3414685.3417776

Alliez, P., Cohen-Steiner, D., Yvinec, M., & Desbrun, M. (2005). Variational tetrahedral meshing. *ACM Trans. Graph.*, *24*(3), 617–625. https://doi.org/10.1145/1073204.1073238

Attene, M. (2020). Indirect predicates for geometric constructions. *Computer-Aided Design*, *126*, 102856. https://doi.org/https://doi.org/10.1016/j.cad.2020.102856

Bridson, R., & Crawford, D. (2014). Quartet: A tetrahedral mesh generator.

Brönnimann, H., Fabri, A., Giezeman, G.-J., Hert, S., Hoffmann, M., Kettner, L., Pion, S., & Schirra, S. (2017). 2D and 3D linear geometry kernel. In *CGAL user and reference manual* (4.11). CGAL Editorial Board. http://doc.cgal.org/4.11/Manual/packages.html#PkgKernel23Summary

Burnikel, C., Mehlhorn, K., & Schirra, S. (1996). *The leda class real number*. Max-Planck-Institut für Informatik. https://books.google.it/books?id=ND5LvwEACAAJ

Cohen-Steiner, D., de Verdière, É. C., & Yvinec, M. (2004). Conforming delaunay triangulations in 3d [Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002]. *Computational Geometry*, *28*(2), 217–233. https://doi.org/https://doi.org/10.1016/j.comgeo.2004.03.001

Diazzi, L., & Attene, M. (2021). Convex polyhedral meshing for robust solid modeling. *ACM Trans. Graph.*, *40*(6). https://doi.org/10.1145/3478513.3480564

Diazzi, L., Panozzo, D., Vaxman, A., & Attene, M. (2023). Constrained delaunay tetrahedrization: A robust and practical approach. *ACM Trans. Graph.*, *42*(6). https://doi.org/10.1145/3618352

Du, Q., & Wang, D. (2003). Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations. *International Journal for Numerical Methods in Engineering*, *56*(9), 1355–1373. https://doi.org/https://doi.org/10.1002/nme.616

Dunyach, M., Vanderhaeghe, D., Barthe, L., & Botsch, M. (2013). Adaptive Remeshing for Real-Time Mesh Deformation. In M. A. Otaduy & O. Sorkine (Eds.), *Eurographics 2013 - short papers*. The Eurographics Association. https://doi.org/10.2312/conf/EG2013/short/029-032

Edelsbrunner, H. (2013). Course notes: Computational geometry and topology.

Edelsbrunner, H., Ablowitz, M. J., Davis, S. H., Hinch, E. J., Iserles, A., Ockendon, J., & Olver, P. J. (2006). *Geometry and topology for mesh generation (cambridge monographs on applied and computational mathematics)*. Cambridge University Press.

Fabri, A., & Pion, S. (2009). Cgal: The computational geometry algorithms library. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. https://api.semanticscholar.org/CorpusID:53246639

Faraj, N., Thiery, J.-M., & Boubekeur, T. (2016). Multi-material adaptive volume remesher. *Computer and Graphics Journal (proc. Shape Modeling International 2016)*, *58*, 150–160.

Freitag, L. A., & Ollivier-Gooch, C. (1997). Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, *40*(21), 3979–4002. https://doi.org/https://doi.org/10.1002/(SICI)1097-0207(19971115)40:21⟨3979::AID-NME251⟩3.0.CO;2-9

George, P., Hecht, F., & Saltel, E. (1991). Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, *92*(3), 269–288. https://doi.org/https://doi.org/10.1016/0045-7825(91)90017-Z

Guan, Z., Song, C., & Gu, Y. (2006). The boundary recovery and sliver elimination algorithms of three-dimensional constrained delaunay triangulation. *International Journal for Numerical Methods in Engineering*, *68*(2), 192–209. https://doi.org/https://doi.org/10.1002/nme.1707

Guibas, L. J., Knuth, D. E., & Sharir, M. (1992). Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica*, *7*(1–6), 381–413. https://doi.org/10.1007/BF01758770

Hu, Y., Schneider, T., Wang, B., Zorin, D., & Panozzo, D. (2020). Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, *39*(4). https://doi.org/10.1145/3386569.3392385

Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., & Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Trans. Graph.*, *37*(4). https://doi.org/10.1145/3197517.3201353

Jacobson, A., Kavan, L., & Sorkine-Hornung, O. (2013). Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, *32*(4). https://doi.org/10.1145/2461912.2461916

Jamin, C., Alliez, P., Yvinec, M., & Boissonnat, J.-D. (2015). Cgalmesh: A generic framework for delaunay mesh generation. *ACM Trans. Math. Softw.*, *41*(4). https://doi.org/10.1145/2699463

Joe, B. (1991). Geompack — a software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, *13*, 325–331. https://api.semanticscholar.org/CorpusID:62639120

Joshi, B. J., & Ourselin, S. (2003). Bsp-assisted constrained tetrahedralization. *International Meshing Roundtable Conference*. https://api.semanticscholar.org/CorpusID:12240850

Karamcheti, V., Li, C., Pechtchanski, I., & Yap, C. (1999). Core library for robust numeric and geometric computation [Proceedings of the 1999 15th Annual Symposium on Computational Geometry ; Conference date: 13-06-1999 Through 16-06-1999]. In *Proceedings of the annual symposium on computational geometry* (pp. 351–359). ACM.

Labelle, F., & Shewchuk, J. R. (2007). Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph.*, *26*(3), 57–es. https://doi.org/10.1145/1276377.1276448

Li, C., Pion, S., & Yap, C. (2005). Recent progress in exact geometric computation [Practical development of exact real number computation]. *The Journal of Logic and Algebraic Programming*, *64*(1), 85–111. https://doi.org/https://doi.org/10.1016/j.jlap.2004.07.006

Murphy, M., Mount, D. M., & Gable, C. W. (2001). A point-placement strategy for conforming delaunay tetrahedralization. *International Journal of Computational Geometry & Applications*, *11*(06), 669–682. https://doi.org/10.1142/S0218195901000699

Rabinovich, M., Poranne, R., Panozzo, D., & Sorkine-Hornung, O. (2017). Scalable locally injective mappings. *ACM Trans. Graph.*, *36*(2). https://doi.org/10.1145/2983621

Ruppert, J. (1995). A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, *18*(3), 548–585. https://doi.org/https://doi.org/10.1006/jagm.1995.1021

Ruppert, J., & Seidel, R. (1992). On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete & Computational Geometry*, *7*(3), 227–253. https://doi.org/10.1007/BF02187840

Sharp, N., Gillespie, M., & Crane, K. (2021). Geometry processing with intrinsic triangulations. *ACM SIGGRAPH 2021 Courses*. https://doi.org/10.1145/3450508.3464592

Shewchuk, J. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, *18*. https://doi.org/https://doi.org/10.1007/PL00009321

Shewchuk, J. (2002). Constrained delaunay tetrahedralizations and provably good boundary recovery. *Proceedings of the 11th International Meshing Roundtable*.

Shewchuk, J. R. (1996, May). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator [From the First ACM Workshop on Applied Computational Geometry]. In M. C. Lin & D. Manocha (Eds.), *Applied computational geometry: Towards geometric engineering* (pp. 203–222, Vol. 1148). Springer-Verlag.

Shewchuk, J. R. (1998). A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, 76–85. https://doi.org/10.1145/276884.276893

Shewchuk, J. R., & Si, H. (2014). Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement. *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, 290–299. https://doi.org/10.1145/2582112.2582138

Si, H. (2015). Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, *41*(2). https://doi.org/10.1145/2629697

Si, H., & Gärtner, K. (2005). Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In B. W. Hanks (Ed.), *Proceedings of the 14th international meshing roundtable* (pp. 147–163). Springer Berlin Heidelberg.

Tournois, J., Wormser, C., Alliez, P., & Desbrun, M. (2009). Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM SIGGRAPH 2009 Papers*. https://doi.org/10.1145/1576246.1531381

Weatherill, N. P., & Hassan, O. (1994). Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, *37*(12), 2005–2039. https://doi.org/https://doi.org/10.1002/nme.1620371203

Zhou, Q., & Jacobson, A. (2016). Thingi10k: A dataset of 10, 000 3d-printing models. *ArXiv*, *abs/1605.04797*. https://api.semanticscholar.org/CorpusID:39867743