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

Saip-Can Hasbay
Student at TU Wien, 01428723

David Hahn
TU Wien, Supervisor

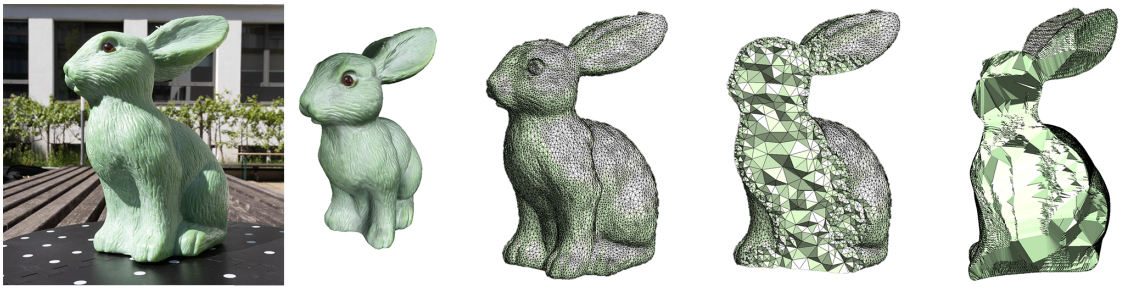

Figure 1: (Left to right) Photograph of a bunny statue; its surface reconstruction using Metashape (AgiSoft, 2022); the exterior and interior of the boundary-approximated tetrahedralization using TetWild (Hu et al., 2018); and the constrained Delaunay tetrahedralization using (Diazzi et al., 2023).

Abstract

We attempt a comprehensive and visual summary of geometry processing, surface, and volumetric meshing. We investigate methods such as Delaunay triangulations, Delaunay tetrahedralizations, intrinsic triangulations. We also review geometry processing in intrinsic triangulations and inverse rendering.

Index Terms: Computing Methodologies—Mesh Generation Meshing—Volume Meshing—Surface Meshing;

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 to the end-users. Meshes can be obtained from 3D reconstruction, 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) (S. Cheng et al., 2016; Hu et al., 2018; Sharp et al., 2021; J. R. Shewchuk, 2002). However, in 3D, although there has been immense work accomplished, robust triangulation or tetrahedralization 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 Tetrahedralizations of a piecewise-linear complex. In this report, we review these works extensively. In summary[1], our contributions are:

- A detailed and visual introduction to Voronoi and Delaunay Diagrams.
- Review of the current state of the art in (constrained) tetrahedral meshing.
- A brief overview of geometry processing in intrinsic triangulations and inverse rendering (please refer to the extended edition (Hasbay, 2024b)).

[1] All of our experiments can be found at (Hasbay, 2024b). In a concurrent report, we also discuss the practical aspects of 3D reconstruction (Hasbay, 2024a).

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

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 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.*

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 the line or 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 that contain the set of points. Intuitively, one can think of a rubber band that tightly snaps a collection of points (Fig. 3).

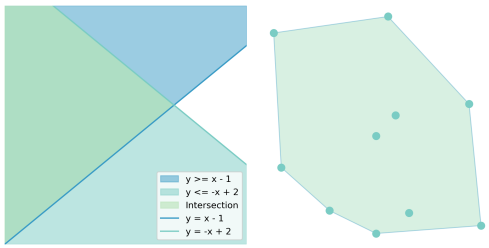

Figure 3: Intersection of two Half-Planes (left) and convex hull (right) of a point set.

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

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

Given a set of points x that 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 of any site of S.

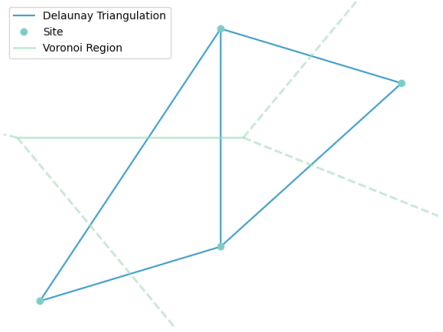

Figure 4: Voronoi diagram and the dual Delaunay triangulation.

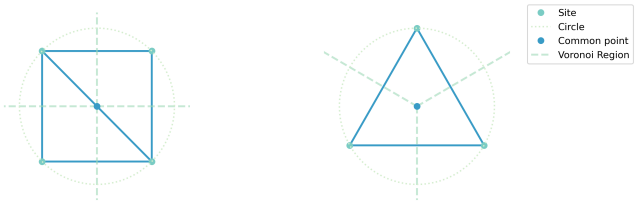

Figure 5: (Left) Four Voronoi edges meet at a common point (i.e., degeneracy). (Right) Three Voronoi edges meet at a common point (i.e., general case).

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. K is the triangulation of S, 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 much literature (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 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 $S \subset \mathbb{R}^2$. This is often a very desirable property: small angles are deleterious for matrix conditioning, but not for interpolation accuracy or discretization error. We omit a further discussion on this matter and refer the reader to (Babuska & Aziz, 1976; J. R. Shewchuk, 2002).

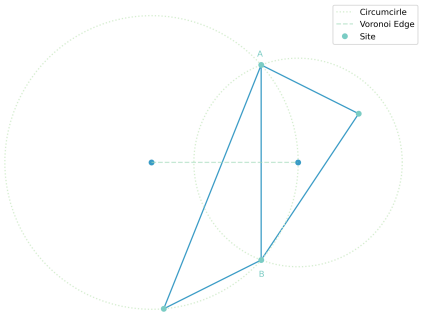

Figure 6: A Delaunay edge ab, and supporting circles.

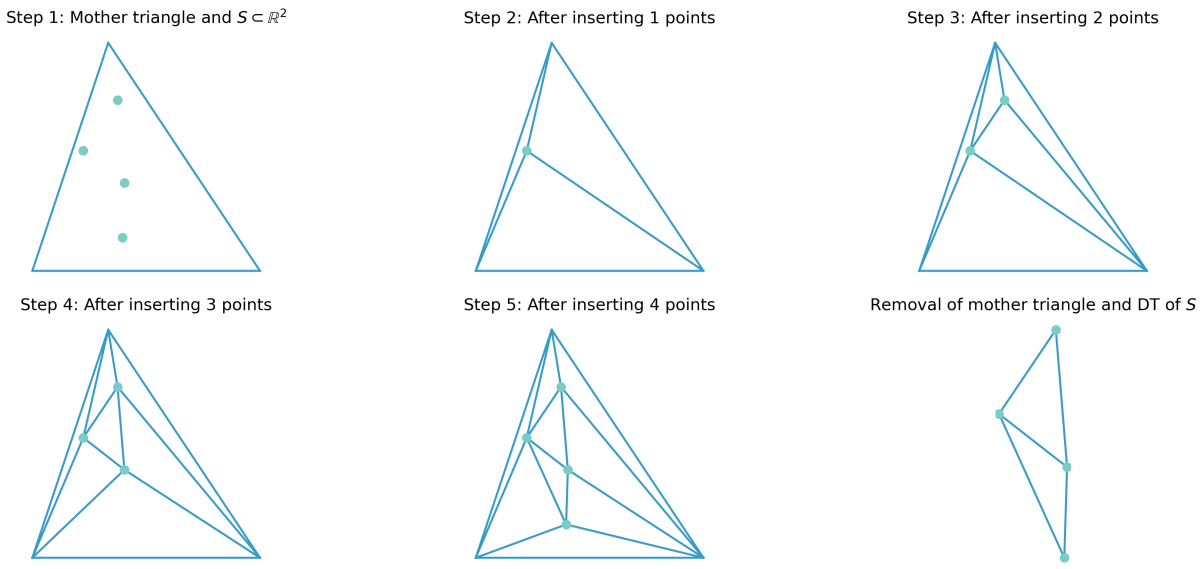

Figure 7: Delaunay triangulation of $S \subset \mathbb{R}^2$. In this example, S consists of 4 sites.

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 ← 1 to n do
    find τ_{i-1} ∈ D_{i-1} containing p_i
    add p_i by splitting τ_{i-1} into three
    while ∃ab not locally Delaunay do
        flip ab to other diagonal cd
    end while
end for
```

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.

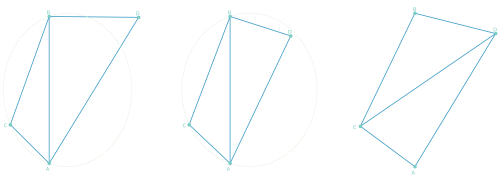

Figure 8: Left: *ab* is locally Delaunay. Middle: *ab* is not locally Delaunay. Right: Result of an edge flip makes *ab* locally Delaunay.

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 (J. R. Shewchuk, 1996a, 1996b). They also use Steiner points for the triangulation (Fig. 9). However, unlike conforming 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.

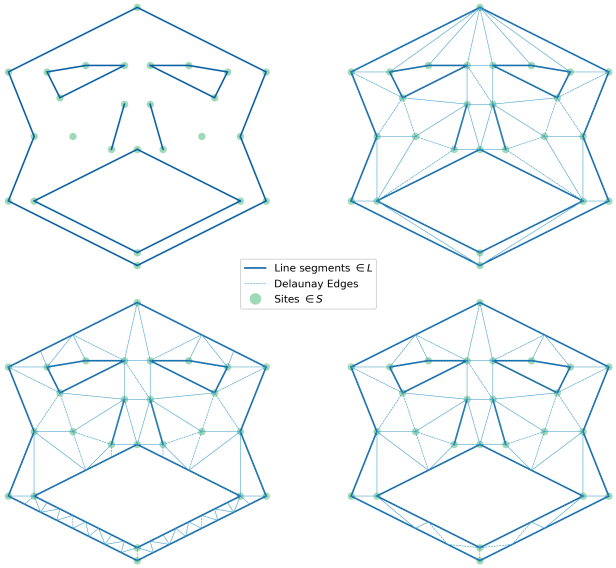

Figure 9: Input (top left), constrained DT (top right), conforming DT (bottom left), and constrained conforming DT (bottom right). We used the *Triangle* library (J. R. Shewchuk, 1996b) for visualization.

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 Tetrahedralizations (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. 10). 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.

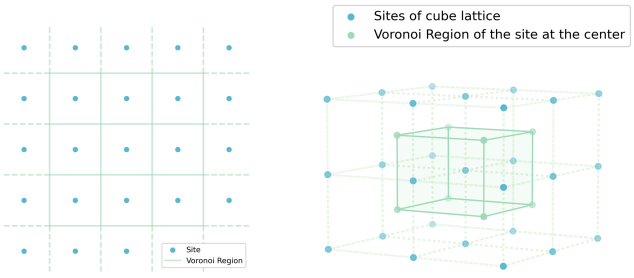

Figure 10: Voronoi diagram of square lattice (left). A subset of sites in the cube lattice, and the Voronoi region (right).

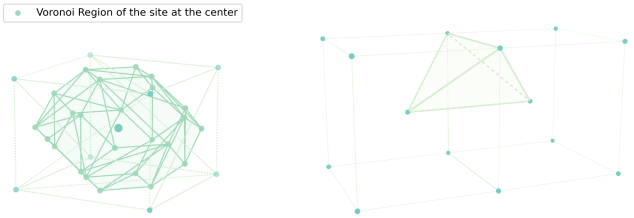

Figure 11: Voronoi region of a site in the BCC lattice (left), and a tetrahedron in the DTET of the BCC lattice (right).

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. 10). 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 intersecting a cube with an octahedron. The resulting shape is a truncated octahedron (Fig. 11). 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. 11). 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. 11). 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 Tetrahedralization. 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 tetrahedralizations. First, we will review boundary-approximating tetrahedralizations, and then move on to the constrained DTETs.

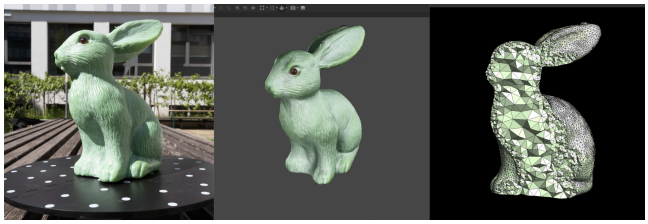

Figure 12: Sapo's bunny found in the *wild* (left), surface reconstruction using Metashape (AgiSoft, 2022) (middle), resulting tetrahedralization (Hu et al., 2018) (right).

3 Approximately Constrained Tetrahedralizations

As the name suggests, boundary-approximating constrained tetrahedralizations 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 tetrahedralizations that require no manual intervention of the input. Rather than expecting clean inputs, they underline 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 tetrahedralization problem. Overall, (Hu et al., 2018) views these key aspects in their approach:

- The input will be more likely imperfect. Deviations from the input within a user-defined envelope ε can be tolerated.
- Reformulate the meshing problem according to the input.
- 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 Tetrahedralizations

We continue with a deep dive into the construction of approximately constrained tetrahedralizations. We start with a problem statement and approach overview, then continue with the method details. 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 (used as $b \times \varepsilon$, where b is the bounding box diagonal of the input) and the desired edge length l. The method aims to construct an *approximately constrained tetrahedralization*. The resulting tetrahedralization of the input triangle mesh must be (1) bounded within the specified ε, and, to ensure an output without inversions (2) it must not contain any inverted elements. Similarly, (3) the edge lengths must be bounded by the desired l. Given that the first and second 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).

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 tetrahedralization, a volumetric Binary Space Partitioning (BSP) tree construction, and barycenter tetrahedralization. The result of this phase is a volumetric mesh that also surrounds the provided model (i.e., tetrahedralization 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 (Bridson & Crawford, 2014; Dey & Levine, 2008; Jamin et al., 2015; Labelle & Shewchuk, 2007; Si, 2015), 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 tetrahedralizations (Sec. 4), boundary-approximating—or, in this case, unconstrained— Delaunay tetrahedralizations 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 convex cells are non-degenerate (i.e., tetrahedra connected to the barycenter

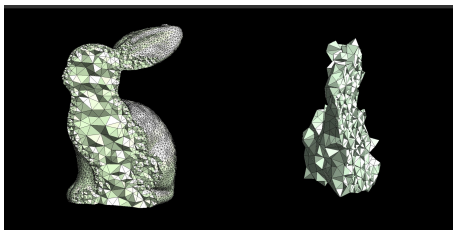

Figure 13: Using the default (0.001) ε value allows us to capture the features of the mesh and increases geometric fidelity (left). Picking an arbitrary high (0.1) value fails to capture features of the input.

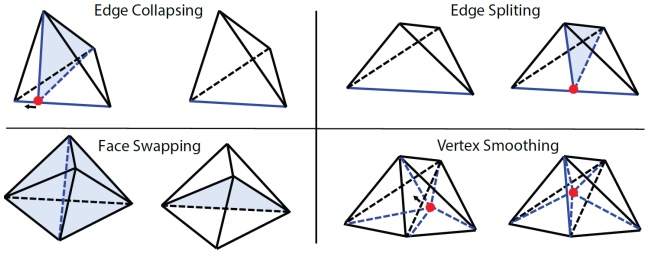

Figure 14: (Hu et al., 2018) employs four local operators for mesh improvement. These are: edge collapsing, edge splitting, face swapping and vertex smoothing. Taken from (Hu et al., 2018).

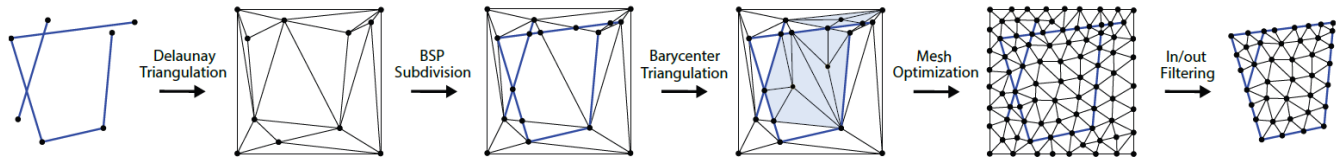

Figure 15: The approach of (Hu et al., 2018) consists of two phases. In the first phase, they generate a valid mesh using exact rationals without hinging on its geometric quality. This phase consists of Delaunay triangulation, BSP-tree construction, and barycenter triangulation. The second phase ensures 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.

will be non-degenerate). The result of this phase is a *valid* 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. For the sake of conciseness, we summarize the most relevant details for our 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; Klingner & Shewchuk, 2007). 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*). 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 (i.e., deviating from ideal regular tetrahedra), 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 due to lack of enough degrees of freedom. The optimization finishes if either the maximum energy is smaller than filtering energy (default filtering energy: less than 10), or the maximum number of iterations is reached (default: 80). Note that the chosen conformal AMIPS energy ranges from 3 to ∞. (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.

3.2 Final Remarks on TetWild

We see no justice in repeating the results of (Hu et al., 2018). Their paper, among plentiful visualizations, 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 tetrahedralization 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 surfaces. 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).

Although boundary-approximating tetrahedralizations proved extremely useful, they rely on lossy and potentially not bijective projections to map boundary conditions (Diazzi et al., 2023). 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 tetrahedralizations, which by design makes them suitable for solving PDEs.

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

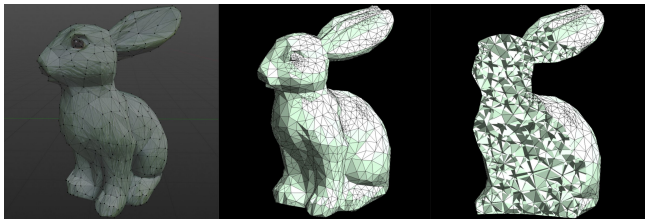

Figure 16: Heavily decimated bunny (left), and its tetrahedralization (Hu et al., 2018) (middle and right).

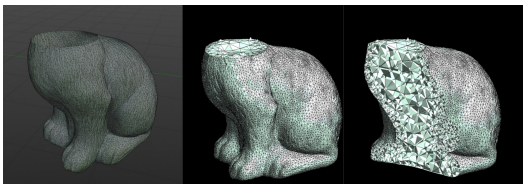

Figure 17: A poor half-eaten bunny (left) and its tetrahedralization (Hu et al., 2018) (middle and right). Even if the input mesh contains holes, it will be closed accordingly.

4 Constrained Delaunay Tetrahedralizations

We now move to constrained Delaunay Tetrahedralizations (DTETs) of a piecewise-linear complex (PLC) (Fig. 18). We will mainly follow the work of (Diazzi et al., 2023). Note that we would have preferred to use the name *conforming, or constrained-conforming* DTETs rather than *constrained* due to the use of Steiner points, but the paper and authors prefer to use *constrained*. Hence, we feel *constrained* to use the word *constrained*. See also (J. Shewchuk, 2002, Sec. 2) for general confusion about the terminology.

Given non-intersecting line segments and vertices, a constrained DT in 2D is always possible (Lee & Lin, 1986; Paul Chew, 1989). 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 Tetrahedralization. Conforming tetrahedralizations (e.g., (Cohen-Steiner et al., 2004; Murphy et al., 2001)) introduce many Steiner points and tetrahedra. 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 (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 tetrahedra 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.

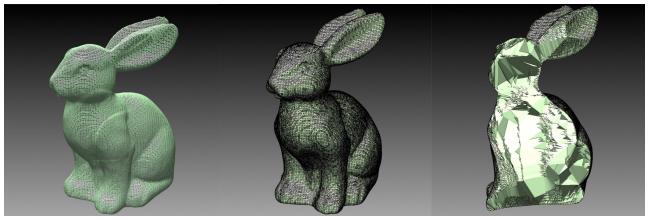

Figure 18: Bunny remeshed as a PLC (left), and resulting constrained DTET (middle, right). Notice the differences in the interior of the resulting tetrahedralization compared to *TetWild* (Fig. 12).

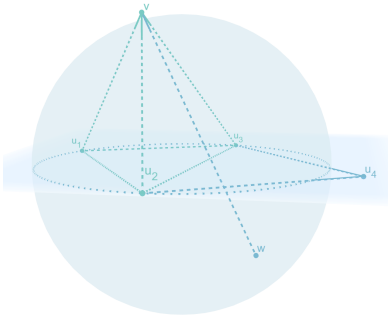

Figure 19: A constrained Delaunay tetrahedron t with vertices u_1, u_2, u_3, v. Vertices v and w belong to opposite sides of the plane, where w is not visible from the interior of t. This plane is in fact the supporting plane to PLC faces u_1, u_2, u_3 and u_2, u_3, u_4. Although u_4 is visible from t, it is outside the circumsphere of t, hence t is constrained Delaunay.

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 constrained DTET 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.

4.1 Constructing Constrained DTETs

We continue with the details of constructing constrained DTETs. We pursue a top-down discussion: first, we define the problem statement. Then provide an overview of (Diazzi et al., 2023) approach. Finally, we discuss their main contributions.

Problem statement. The input is assumed to be a piecewise-linear complex (PLC) (Miller, 1998). A PLC consists of vertices, segments, and polygonal facets. The task is to compute a constrained DTET that preserves the original PLC. More specifically, (1) the vertices of resulting tetrahedralization T must only contain the vertices of the original PLC P. Similarly, (2) each facet of P must be the union of some triangular facets of T. Moreover, (3) T is constrained Delaunay to P. Consequently, the circumsphere that bounds each tetrahedron contains no visible vertex of P (Fig. 19).

Overview: Robust constrained DTET. In brief terms, (Diazzi et al., 2023) pursue the following procedure for a robust constrained DTET:

1. Perform standard Delaunay tetrahedralization of the input vertices based on incremental insertion.
2. Perform robust segment recovery using implicit point type.
3. Perform robust face recovery using novel numerical kernel approach.
4. Perform internal-external characterization of the resulting tetrahedra.

Steps 2 and 3 are the main contribution of (Diazzi et al., 2023). But, they also discuss the expanding cavity issue.

Characteristics of a PLC. Not all PLCs admit a constrained DTET. Assuming the segments of a PLC are strongly Delaunay, then the PLC admits a constrained DTET. A PLC is strongly Delaunay, if it admits a circumsphere that does not contain or touch other PLC vertices. Hence, if we split PLC segments at the correct points, we obtain a PLC that admits a constrained DTET. These so-called correct split points are called Steiner points. (Diazzi et al., 2023) use the splitting method introduced by (Si & Gärtner, 2005) due to resulting fewer Steiner points.

Segment recovery. Assuming that a PLC segment is not in the DT, it is called a missing segment. The main idea is to split the missing segment into shorter sub-segments until it satisfies the condition. Beware that the split point may be determined by the intersection of a segment with a sphere. This implies that its coordinates can be irrational numbers. We refer the reader to (Diazzi et al., 2023) for a more concrete discussion of segment recovery.

Face recovery. Assuming that all the PLC segments are in the DT, Si's method checks if all PLC faces consist of the union of Delaunay triangles. If this verification fails for a PLC face, then it is denoted as *missing*. Given a missing face, then it must be pierced by at least one edge of the tetrahedralization. To recover the missing face, the region around the face must be retetrahedralized (Fig. 20). This is done by (1) splitting the cavity along the missing face. (2) Then, for each cavity, computing a local DT to fill the cavity with the new tetrahedra. (3) Lastly, combining local DTs to obtain the final retetrahedralization. Note that if we cannot compute a local DT (2), this approach might fail, which we discuss next.

Potential failures in Tetgen. Overall, Si's method is efficient. But, it has two disadvantages. First, Steiner points may have irrational coordinates. Hence, a numerically robust implementation requires predicates and data types that can deal with irrational numbers. Second, the cavity expansion process described might fail. Si's method assumes that local tetrahedralizations of the two half-cavities have disjoint interiors and that their boundaries match at the face being recovered (Diazzi et al., 2023). However, the expansion process may add a new tetrahedron, which may lead to an intersection of two tetrahedralizations (Fig. 21). (Diazzi et al., 2023) mention that this case occurred only twice from the 4408 models that they tested. They conclude that this problem cannot be overlooked.

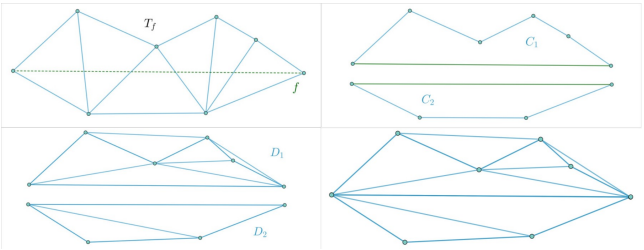

Figure 20: Face recovery in 2D. First, (Diazzi et al., 2023) intersects the missing face f by mesh triangles T_f (top left). Then, they separate f to create two half-cavities C_1, C_2 (top right). Each half-cavity is then Delaunay-triangulated D_1, D_2 (bottom left). Lastly, the triangles outside of C_1, C_2 are removed and cavities are merged whilst preserving f (bottom right).

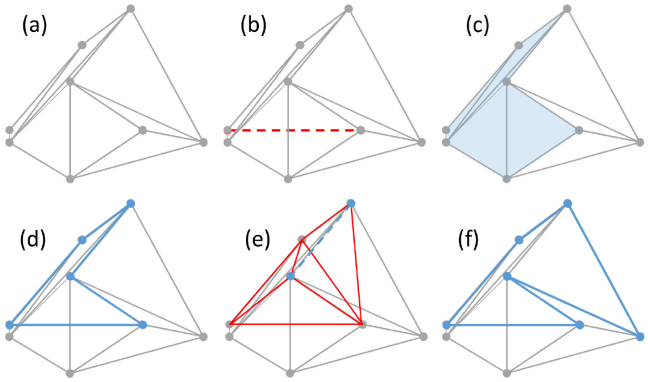

Figure 21: Example of the cavity expansion failure in 2D. Note that 3D face recovery corresponds to 2D segment recovery. (a) Having recovered any segment before, the Delaunay property of the vertices may not hold anymore. (b) A missing segment, (c) its corresponding cavity, and (d) the upper-half cavity. (e) The DT of the half-cavity (red) has a missing segment (blue). (f) Hence, the algorithm expands across it, which results in a half-cavity that intersects with the lower half-cavity. Figure is taken from (Diazzi et al., 2023).

Tackling numerical robustness. The easiest solution to tackle the numerical robustness is to use exact arithmetic kernels. The CORE (Karamcheti et al., 1999) library is a good candidate for exact arithmetic kernels. Combining it with the lazy evaluation paradigm employed by CGAL (Fabri & Pion, 2009) improves efficiency. Nevertheless, this approach is still impractical—hours for small input files.

Rational Steiner points to the rescue. To address the performance drawback of the CORE library, (Diazzi et al., 2023) opts for using (arbitrarily precise) rational numbers. They found that Steiner points must be placed exactly on the segment they split. If the position is *snapped* to an approximated location, then the input PLC is deformed. This leads to non-robust implementations that might result in a crash. Given the approximated Steiner point remains along the segment, it can be (re)positioned without changing the mesh connectivity. (Diazzi et al., 2023) computes approximated Steiner point by a linear combination of segment vertices: $tv_1 + (1-t)v_2$, where $t \in (0,1)$. This approach leads to using faster number types such as GNU GMP rationals (Granlund, 1996).

Implicit Steiner points. By using custom indirect geometric predicates (Attene, 2020), (Diazzi et al., 2023) replaces GNU GMP for improved efficiency. Indirect predicates are evaluated using standard floating-point calculations. They are used to retrieve the exact mutual position of *implicit points*. An implicit point is an un-evaluated expression that represents a position in space. Examples of implicit points are line-plane intersection (LPI) and three-plane intersection (TPI). An *explicit* point is a point with known floating point coordinates. LPI can be seen as a function with five explicit points—two for the line and three for the plane of intersection. Using the linear combination of two explicit points (v_1, v_2), (Diazzi et al., 2023) represent Steiner points implicitly. Consequently, they call the expression $tv_1 + (1-t)v_2$ a linear combination of an implicit point. We leave the implementation details to the reader (Diazzi et al., 2023).

Tackling the cavity expansion issue. To address the cavity issue, (Diazzi et al., 2023) employs an approach based on (J. R. Shewchuk, 2000) (Fig. 22). (1) Using a local 2D constrained DT each face is triangulated. (2) To form a tetrahedron, each triangle is connected to one *apex* vertex. (3) If only one vertex in the set is a valid apex for any given triangle, then the constrained DTET is unique. Therefore, assuming a valid apex is chosen, the resulting tetrahedron is guaranteed to be part of the constrained DTET.

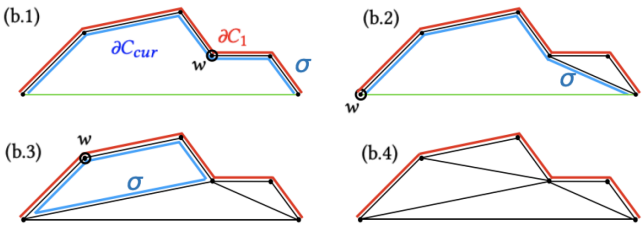

Figure 22: Example of gift wrapping in 2D. Remember the missing PLC segment f (Fig. 20) and how we get two half-cavities. (Diazzi et al., 2023) triangulate the upper cavity (b.1-b.4) using the gift-wrapping algorithm. The cavity boundary ∂C_1 remains constant, but, ∂C_{cur} does not. At each step an edge σ of ∂C_{cur} is connected to an apex w to create a valid triangle. From (Diazzi et al., 2023).

(Diazzi et al., 2023) combines this gift-wrapping algorithm with a coherent symbolic perturbation technique (Edelsbrunner & Mücke, 1990). Even though the points might not be in general position, this perturbation technique assures conformity. To summarize (Diazzi et al., 2023)'s approach: (1) Sequentially construct cavity. (2) Split it into two half-cavities. (3) For each half-cavity, sequentially build one tetrahedron. Throughout this process, (Diazzi et al., 2023) consider three key aspects:

- All predicates and checks must be exact due to the errors that stem from numerical imprecision.
- The resulting tetrahedralization of two half-cavities must match on the common face, even though the vertices might not be in general position.
- The shared triangles of the half-cavities are unknown during tetrahedralization. The implementation must ensure consistency of the alignment of the two halves.

To finalize, (Diazzi et al., 2023) certify the efficiency of their modified gift-wrapping algorithm. However, they note that (Si & Gärtner, 2005)'s approach is faster in practice. Consequently, the modified gift-wrapping algorithm is only used, if the cavity expansion fails. Having discussed the main contributions of (Diazzi et al., 2023), we will now make some final remarks.

4.2 Concluding Remarks on Constrained DTETs

We briefly make some final remarks on constrained DTETs. There is no point repeating the results of (Diazzi et al., 2023) except the essential parts. The interested reader should refer to (Diazzi et al., 2023) for a thorough analysis and comparison against (Bridson & Crawford, 2014; S.-W. Cheng et al., 2008; Diazzi & Attene, 2021; Hu et al., 2018; Rineau & Yvinec, 2007; Si & Gärtner, 2005).

Overall, the results of (Diazzi et al., 2023) indicate a fast approach on a Linux-based machine equipped with an AMD EPYC 7452 CPU and 1Tb RAM. They tested their implementation on the meshes of the Thingi10k (Zhou & Jacobson, 2016) that fulfill the PLC condition (i.e., in 4408 models with no self-intersections). Roughly 98% of the models have an average execution time of 1.8 seconds. The average peak memory allocation is noted to be 46.5 Mb. On average, about 25k Steiner points are used.

(Diazzi et al., 2023) note that the resulting tetrahedral mesh does not conform to the *connectivity* of the original PLC due to edge-based Steiner points. The findings of (Alexa, 2019) indicate that 3D Delaunay condition does not always lead to high-quality meshes. This also implies that the results of (Diazzi et al., 2023) may not always give high-quality results. They also note that their approach does not contain a mesh optimization phase. Furthermore, due to the use of floating-point arithmetic in Delaunay refinement methods, they argue that the result might not guarantee convergence. Therefore, using their approach to initialize refinement methods does not guarantee success. To ensure convergence, the authors mention that utilizing new implicit points and indirect predicates would be necessary, which they see as a very interesting direction for future research. Similarly, they note that conversion of implicit points to floating points without flips remains an unsolved problem. Lastly, they state the necessity of future research on the conditions that allow a PLC to admit a floating point representable constrained DTET.

5 Conclusion

In this report, we discussed Delaunay triangulations and tetrahedralizations. We began our discussion with Voronoi and Delaunay Diagrams in 2D. Next, we discussed more complicated matters such as constrained and conforming Delaunay triangulations. We observed as we extended the 2D case into 3D, the concepts of Delaunay remained mathematically robust. However, more likely due to our feeble brains—at least for this author—we realized, even after years of research, that the 3D case is not as intuitive as the 2D. The usability of (constrained, conforming, *...what is next? ...*) Delaunay tetrahedralizations in practical scenarios remains not in its final form, and this statement is supported by the sheer amount of work published in this field.

Most notably, we discussed the recent work of (Hu et al., 2018) (*TetWild*) and (Diazzi et al., 2023) (*CDT*). While we struggled in the trenches, we saw that at least the approach of (Hu et al., 2018) shed some light on the front for boundary approximating tetrahedralizations. Their approach is not only robust but also simple and user-friendly with only two main parameters to tweak. And, as of lately, arguably one of their biggest drawback, performance, seems to be addressed with the *Fast TetWild* (Hu et al., 2020). Similarly, arguably the work of (Diazzi et al., 2023) comes to the rescue for constrained Delaunay tetrahedralizations on piecewise-linear complices. Although their work possibly addresses a more specific field (e.g., applications in engineering, or architecture), their robust implementation opens another door for future research. The work of (Diazzi et al., 2023) especially underlined the necessity of robust boundary-preserving tetrahedral meshes for solving partial differential equations (PDEs). Once again, we see that one of the overarching goals is to solve PDEs. Maybe having an *intrinsic* outlook on this problem would help to accelerate the progress in this field.

As geometric data becomes more present—not necessarily in good quality—in a variety of fields, algorithms need to process low-quality geometric data reliably. In 1900, the renowned physicist Lord Kelvin fell victim to his own confidence with the claim: "There is nothing new to be discovered in physics now. All that remains is more and more precise measurement"(Tyson, 2014). Five years later, Albert Einstein proved him wrong with his special theory of relativity, and the rest is history. The development of the intrinsic—rather than extrinsic—point of view was a turning point in modern mathematics and physics, which led to Einstein and Hilbert's development of the theory of general relativity (Sharp et al., 2021). Most recently, the work of (Sharp et al., 2019, 2021) in intrinsic triangulations looks at geometry processing in a new light. Although their work, as far as we know, is only limited to surface meshes, they indicate the possibility of extending the intrinsic point of view onto volumetric meshes. Before we conclude, we note that in the extended edition of this report ToDo: *** add link ***, we review the work of (Sharp et al., 2021) for intrinsic triangulations, as well as another intriguing work that utilizes inverse rendering for geometry processing (Nicolet et al., 2021). With that, we conclude by hoping that we analyzed the current state of the art adequately and that it was not only helpful for this grumbling author but also at some level for the reader.

Acknowledgments

This work is dedicated to Alyn Wallace, who unfortunately passed away at a very young age quite recently. His enthusiasm and tutorials for astronomy and astrophotography influenced many immensely including this author. We also greatly thank David Hahn who once again had enough patience to deal with this grumbling author.

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

Babuska, I., & Aziz, A. K. (1976). On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, *13*, 214–226. https://api.semanticscholar.org/CorpusID:123988912

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

Cheng, S.-W., Dey, T. K., & Levine, J. A. (2008). A practical delaunay meshing algorithm for a large class of domains. *Proceedings of the 16th International Meshing Roundtable*, 477–494.

Cheng, S., Dey, T., & Shewchuk, J. (2016). *Delaunay mesh generation* [ISBN 9781584887317]. CRC Press. https://books.google.at/books?id=oJ3SBQAAQBAJ

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

Dey, T. K., & Levine, J. A. (2008). Delpsc: A delaunay mesher for piecewise smooth complexes. *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*, 220–221. https://doi.org/10.1145/1377676.1377712

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)* [ISBN 052168207X]. Cambridge University Press.

Edelsbrunner, H., & Mücke, E. P. (1990). Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (tog)*, *9*(1), 66–104.

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

Granlund, T. (1996). Gnu mp: The gnu multiple precision arithmetic library. *http://gmplib.org/*.

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

Hasbay, S.-C. (2024a). 3D Vision: Reconstruction Project [Accessed: 2024-06-07]. https://github.com/sapo17/3DVision/blob/main/Hasbay_01428723_3DVisionReport.pdf

Hasbay, S.-C. (2024b). Seminar in Computer Graphics (S2024) [Accessed: 2024-06-07]. https://github.com/sapo17/GraphicsSeminarSS24

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.

Klingner, B. M., & Shewchuk, J. R. (2007). Agressive tetrahedral mesh improvement. *Proceedings of the 16th International Meshing Roundtable*, 3–23. http://graphics.cs.berkeley.edu/papers/Klingner-ATM-2007-10/

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

Lee, D. T., & Lin, A. K. (1986). Generalized delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, *1*(3), 201–217. https://doi.org/10.1007/BF02187695

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

Miller, G. L. (1998). Control volume meshes using sphere packing. *International Symposium on Solving Irregularly Structured Problems in Parallel*, 128–131.

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

Nicolet, B., Jacobson, A., & Jakob, W. (2021). Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, *40*(6). https://doi.org/10.1145/3478513.3480501

Paul Chew, L. (1989). Constrained delaunay triangulations. *Algorithmica*, *4*(1–4), 97–108. https://doi.org/10.1007/BF01553881

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

Rineau, L., & Yvinec, M. (2007). A generic software design for delaunay refinement meshing. *Comput. Geom. Theory Appl.*, *38*(1–2), 100–110. https://doi.org/10.1016/j.comgeo.2006.11.008

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

Sharp, N., Soliman, Y., & Crane, K. (2019). Navigating intrinsic triangulations. *ACM Trans. Graph.*, *38*(4). https://doi.org/10.1145/3306346.3322979

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. (1996a). Triangle: Definitions of Geometric Terms [https://shorturl.at/rvxyI].

Shewchuk, J. R. (1996b, 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. (2000). Sweep algorithms for constructing higher-dimensional constrained delaunay triangulations. *Proceedings of the sixteenth annual symposium on Computational geometry*, 350–359.

Shewchuk, J. R. (2002). What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures. https://api.semanticscholar.org/CorpusID:15379771

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

Tyson, N. D. (2014, September). *Death by black hole* [ISBN 9780393350388 (ISBN10: 039335038X)]. WW Norton.

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