Weierstraß-Institut

für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e. V.

Preprint ISSN 0946 – 8633

3D boundary recovery by constrained Delaunay tetrahedralization

Hang Si[1], Klaus Gärtner[1]

submitted: August 5, 2010

[1] Weierstraß-Institut für
Angewandte Analysis und Stochastics
Mohrenstr. 39, 10117 Berlin
E-Mail: Hang.Si@wias-berlin.de
E-Mail: Klaus.Gaertner@wias-berlin.de

No. 1530
Berlin 2010

2010 *Mathematics Subject Classification.* 52B55, 65D18.

Key words and phrases. tetrahedral mesh generation, boundary recovery, constrained Delaunay tetrahedralization, Steiner points.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 2044975
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Abstract

Three-dimensional boundary recovery is a fundamental problem in mesh generation. In this paper, we propose a practical algorithm for solving this problem. Our algorithm is based on the construction of a *constrained Delaunay tetrahedralization* (CDT) for a set of constraints (segments and facets). The algorithm adds additional points (so-called *Steiner points*) on segments only. The Steiner points are chosen in such a way that the resulting subsegments are Delaunay and their lengths are not unnecessarily short. It is theoretically guaranteed that the facets can be recovered without using Steiner points. The complexity of this algorithm is analyzed. The proposed algorithm has been implemented. Its performance is reported through various application examples.

1. INTRODUCTION

Given a three-dimensional polyhedron P, a fundamental problem is to form a "coarse"partition of P by a set of simple cells such that the boundary of P is represented by the cells. By coarse we mean that the number of cells is small. Many applications are based on it.

This problem has many difficulties. It is known that additional points (so-called *Steiner points*) are necessary in order to form a tetrahedralization of a polyhedron [1,2]. Two examples are shown in Fig. 1. Moreover, the problem to determine whether a simple polyhedron can be tetrahedralized without Steiner points is NP-complete [3]. There are polyhedra which may require a large number of Steiner points to be tetrahedralized [4], see Fig. [1] right. One challenging question is: Given any polyhedron, can we tetrahedralize it with the number of Steiner points as small as possible?

This problem has been long addressed in mesh generation methods. The most popular methods are proposed in [5, 6, 7].

A common theme of these methods is: At first, an initial Delaunay triangulation of the vertex set of a polyhedron P is constructed. Next, the boundary

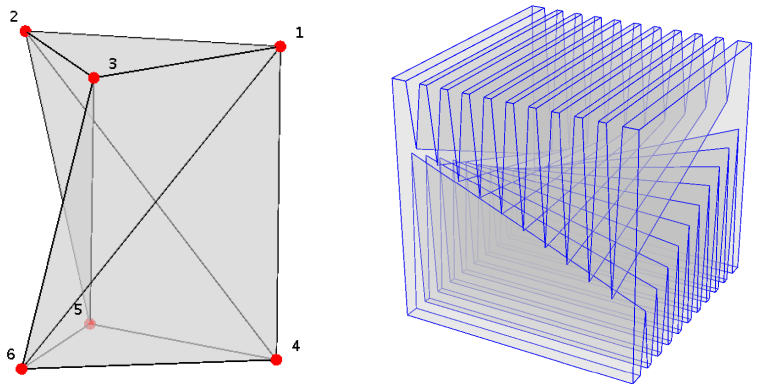


Figure 1. Two polyhedra which are not tetrahedralizable without Steiner point. Left: The Schönhardt polyhedron [2] can be obtained by twisting the upper face around the axes of a parallel triangular prism by a small angle. Right: The Chazelle's polyhedron [4] formed by cutting wedges from a cube. In the middle of the polyhedron are two sets of orthogonal lines. The lower and upper lines lie on hyperbolic paraboloids $z = xy$, and $z = xy + \epsilon$, respectively.

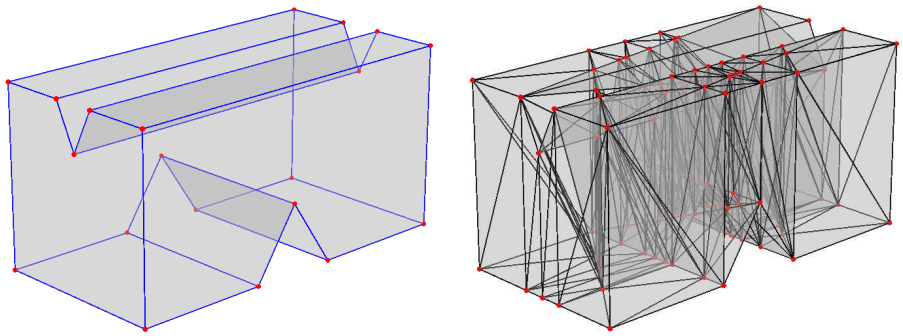
of P will be recovered by modifying the triangulation (e.g., edge/face swaps), Steiner points are added when they are needed. They are efficient in engineering applications. However, many operations, such as the edge/face swaps and the locations of Steiner points, are still not well studied. These methods usually rely on *ad hoc* techniques to address questions like "how to perform combined swaps?" and "where to place Steiner points?". The implementation of these methods is generally a complex task. Their complexities are hard to analyze.

Convex decomposition is a theoretical problem studied in computational geometry [4, 8, 9, 10, 11, 12]. Chazelle showed that any simple polyhedron (which contains no holes) P of n vertices can be partitioned into $O(n^2)$ tetrahedra [4]. Chazelle and Palios [8] improved this upper bound to $O(n + r^2)$, where r is the number of reflex edges of P (an edge of P *reflex* if its adjacent faces form an angle larger than 180^o). These works established the complexity of the problem. However, they are neither general nor practical. In fact, the algorithm in [8] may result in a large number of output tetrahedra, see Fig. 2 (b).

Delaunay triangulations are well-studied objects, see, e.g., [13, 14]. A simple systematic approach to recover boundaries is to enrich the vertex set V of a polyhedron P by adding Steiner points into the boundary of P until the Delaunay triangulation for V respects the boundary of P [15, 16, 17]. The result is a so-called *conforming Delaunay triangulation*, see Fig. 2 (c). This approach, however, may result in an unnecessarily large number of Steiner points, especially when the boundary of the polyhedron contain small angles.

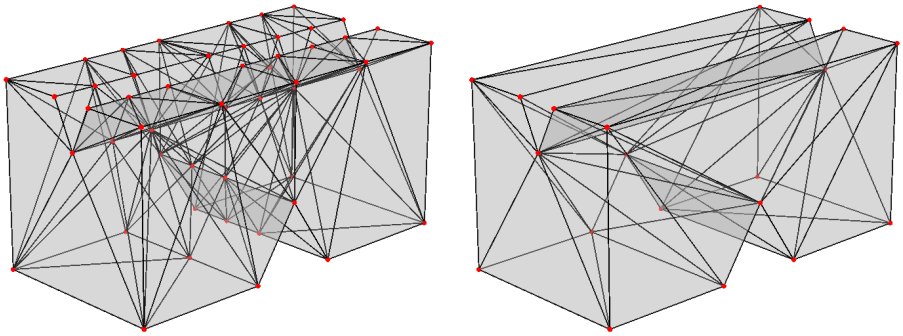
An alternative approach for this problem is to construct a *constrained Delaunay triangulation* [18, 19, 20, 21], which is a triangulation for a point set that respects a set of constraints (edges and faces), and it has properties close to those of a Delaunay triangulation, see Fig. 3. This approach has both theoretical and practical features. It usually needs less Steiner points than the conforming Delaunay triangulation approach. The termination of this approach can be theoretically proved. It is possible to analyze its complexity.

We comment that constrained Delaunay triangulations are useful objects in other applications as well. Many finite element mesh generation methods [22, 23, 24] use them



(a) The input polyhedron
20 nodes, 12 facets

(b) Convex decomposition
138 nodes, 280 tets



(c) Conforming Delaunay
51 nodes, 103 tets

(d) Constrained Delaunay
20 nodes, 29 tets

Figure 2. Examples of different approaches for meshing a simple polyhedra.

as the intermediate meshes to generate good quality tetrahedral meshes suitable for numerical simulation. Since the boundaries are respected, many useful informations of the mesh domain, such as the nearest boundary, the local feature size, etc, can be efficiently obtained from them. They are perfect intermediate objects for generating conforming Delaunay meshes [25, 26].

Algorithms for generating constrained Delaunay tetrahedralizations are proposed in [20, 27, 21]. A key question in these algorithms is how to choose Steiner points. In [20], the majority of Steiner points are chosen at the midpoint of missing boundary edges. This algorithm may create unnecessary Steiner points. A set of Steiner points insertion rules are introduced in [21], such that the Steiner points are chosen adaptively according to the geometric neighbor informations. These choices of Steiner points considerably reduced the number of Steiner points.

Once all boundary edges of P are recovered, the next key question is how to recover the facets of P? Shewchuk shows that it is possible to recover the facets without using Steiner points [28]. So far, Shewchuk has proposed several algorithms for this purpose [20, 27]. The flip-based facet insertion algorithm [27] has a better complexity. In [21], a simple facet recovery algorithm based on local Delaunay tetrahedralizations is proposed.

Some applications in mesh generation, e.g., local re-meshing, use a pre-discretized surface mesh as input, and require that the tetrahedral mesh of the domain must match the surface

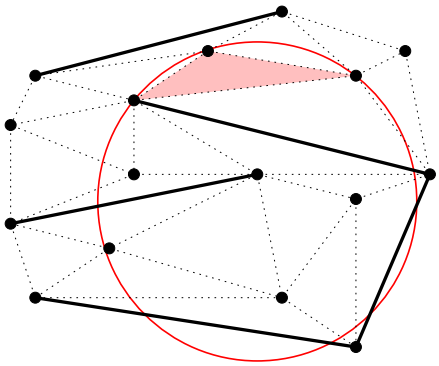


Figure 3. Constrained Delaunay triangulation. The circumcircle of the shaded triangle is not empty but no enclosed vertex is visible from its interior.

mesh exactly. This requirement imposes a stronger restriction to this problem, i.e., no Steiner points are added on the surface triangulation. Hence Steiner points are only added into the interior of the object. Since a constrained Delaunay triangulation may contain Steiner points in its boundary, hence the above algorithms do not directly solve this problem. We propose to solve it by a post-processing step which removes the Steiner points from the boundary.

In this paper, we present a practical algorithm for constructing constrained Delaunay tetrahedralizations of arbitrary polyhedra. This algorithm is based on our previous work [21] and it contains many substantial improvements. The presented algorithm greatly simplifies the original one and is much more efficient. The phase of local degeneracy removal in [21] can be completely skipped. The complexity of the algorithm is analyzed.

Outline The paper is organized as follows. We first formalize the meshing problem in Section 2. In Section 3 we give a definition of constrained Delaunay tetrahedralizations and show some of their basic properties. The proposed algorithm is described in Section 4. The individual phases (segment recovery and facet recovery) of the algorithm are discussed in the Section 5 and Section 6, respectively. The analysis of the algorithms are given therein. We discuss the approach for deleting and suppressing Steiner points in Section 7. Some experimental results from publicly available examples are shown in Section 8. A conclusion and an outlook is given in Section 9.

2. THE MESHING PROBLEM

In this section, we formalize the meshing problem treated by the presented algorithm. We first define the input objects. A *physical domain* Ω in $\mathbb{R}^3$ is the volume enclosed by its *boundary* $\partial\Omega$. Usually, $\partial\Omega$ may be arbitrarily shaped, e.g., curved edges and surfaces. Many applications require that $\partial\Omega$ includes *internal boundaries* which separate Ω into sub-domains so that different materials can be modeled. Hence Ω is generally not a topological manifold. A *mesh domain* is an object which approximates Ω topologically and geometrically.

We define a not necessarily convex *polyhedron* as the union of convex polyhedra, $P = \bigcup \mathcal{P}$,

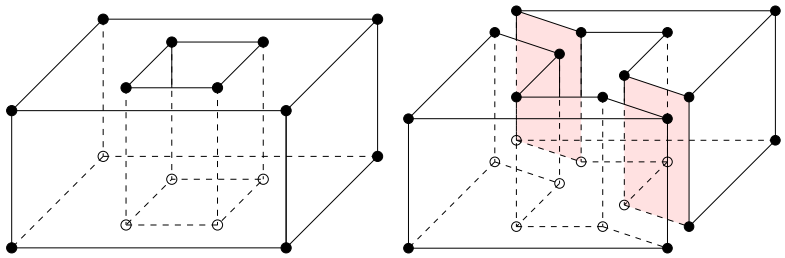


Figure 4. Polyhedra and faces. Left: A polyhedron (a handle) formed by the union of four convex polyhedra. Two faces (one at top and one at bottom) of it are not simply connected. Right: Two polyhedra. The shaded area highlights two 2-faces whose points have the same face figures.

where $\mathcal{P}$ is a finite set of convex polyhedra, and the space of P is connected, see Fig. 4. The *dimension* of P, $\dim(P)$, is the largest dimension of a convex polyhedron in $\mathcal{P}$.

The definition of a *face* of a polyhedron P is taken from Edelsbrunner [29] with an additional requirement in its connectness. Let $\mathbb{B}_\epsilon$ be the open ball of radius ϵ centered at the origin in $\mathbb{R}^d$. For a point $\mathbf{x} \in \mathbb{R}^d$ we consider a sufficiently small neighborhood $N_\epsilon(\mathbf{x}) = (\mathbf{x} + \mathbb{B}_\epsilon) \cap P$. The *face figure* of $\mathbf{x}$ is the enlarged version of this neighborhood within this polyhedron, i.e., $\mathbf{x} + \bigcup_{\lambda>0} \lambda(N_\epsilon(\mathbf{x}) - \mathbf{x})$. A *face* of P is the closure of a maximal connected collection of points with identical face figures. By this definition, a face F of P may contain holes, but the space of F must be connected, see Fig. 4 for examples. F is again a polyhedron. The dimension of F is the dimension of its *affine subspace* $\mathrm{aff}(F)$, i.e., $\dim(F) = \dim(\mathrm{aff}(F))$. A 0-face is a vertex, a 1-face is an edge (also called a *segment*), and a $(\dim(P)-1)$-face is called a *facet* of P. All proper faces of P form the *boundary* $\mathrm{bd}(P)$ of P. The *interior* of P is $\mathrm{int}(P) = P - \mathrm{bd}(P)$.

We define a *piecewise linear system* (abbreviated as PLS) to be a finite collection $\mathcal{X}$ of polyhedra with the following properties

(i) $P \in \mathcal{X} \Longrightarrow$ all faces of P are in $\mathcal{X}$,
(ii) $P, Q \in \mathcal{X} \Longrightarrow \exists K \in \mathcal{X}$ s. t. $P \cap Q = \cup_{K \in \mathcal{X}} K$, and
(iii) $\dim(P \cap Q) = \dim(P) \Longrightarrow P \subseteq Q$ and $\dim(P) < \dim(Q)$.

This definition generalizes the one introduced by Miller *et al.* [30] by allowing non-convex polyhedra, see Fig. 5. PLSs are flexible for representing non-manifold objects. The properties *(i)* and *(ii)* are essential, they ensures that a PLS is closed by both taking boundaries and taking intersections. The property *(ii)* is relaxed from that of a complex. For example, an edge and a quadrilateral may intersect at a point $\mathbf{v}$ as long as $\mathbf{v} \in \mathcal{X}$. Since two non-convex polyhedra P and Q may intersect at more than one faces of them, $P \cap Q$ is a union of elements of $\mathcal{X}$. *(iii)* is an extra property which makes a PLS more flexible. For example, it allows a cube to enclose an edge in its interior without the need of further decomposition. Furthermore, it excludes the case of two polyhedra having the same dimension to overlap each other.

The dimension of a PLS $\mathcal{X}$, $\dim(\mathcal{X})$, is the largest dimension of its polyhedron. A *subsystem* of $\mathcal{X}$ is a subset of $\mathcal{X}$ which is again a PLS. A particular subsystem is the *i-skeleton*, $\mathcal{X}^{(i)}$, of $\mathcal{X}$ which consists of all polyhedra of $\mathcal{X}$ whose dimensions $\leq i$. For example, $\mathcal{X}^{(0)}$ is the *vertex set*, $\mathrm{vert}(\mathcal{X})$, of $\mathcal{X}$. The *boundary system*, $\partial\mathcal{X}$, of $\mathcal{X}$ is the $(\dim(\mathcal{X})-1)$-skeleton of $\mathcal{X}$. The *underlying space* of $\mathcal{X}$ is $|\mathcal{X}| = \bigcup_{P \in \mathcal{X}} P$. Note that $|\mathcal{X}| \subseteq \mathbb{R}^d$ is a topological subspace of $\mathbb{R}^d$. The collection $\mathcal{X}$ gives a special topology on $|\mathcal{X}|$, refer to [26].

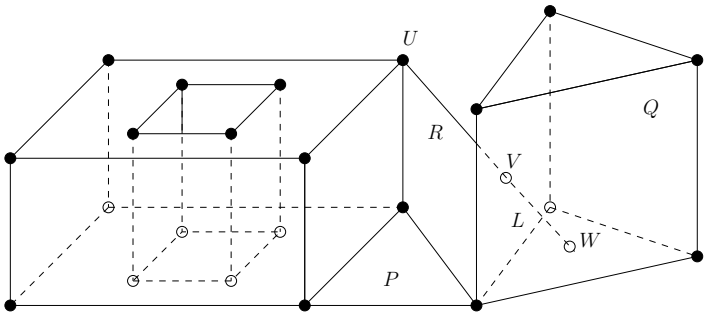


Figure 5. A three-dimensional piecewise linear system $\mathcal{X}$. In this figure, the dimensions of polytopes $Q, P, R, L \in \mathcal{X}$ are $3, 2, 1, 1$, respectively. $U, V, W \in \mathcal{X}$ are 0-polytopes, where $V = R \cap L = R \cap Q$, $L = L \cap Q$, and $\dim(L) < \dim(Q)$.

Given a physical domain Ω, we use a PLS $\mathcal{X}$ to represent it such that Ω and $|\mathcal{X}|$ are homeomorphic (i.e., they are topologically equivalent) and the shape of Ω is approximated by $|\mathcal{X}|$ geometrically.

Next we define the output objects. A *triangulation* of a PLS $\mathcal{X}$ is a simplicial complex $\mathcal{T}$ such that the underlying space of $\mathcal{T}$ equals to the convex hull of the vertices of $\mathcal{X}$ and every polyhedron of $\mathcal{X}$ is represented by a subcomplex of $\mathcal{T}$. More formally, $\mathcal{T}$ satisfies

(i) $|\mathcal{T}| = \text{conv}(\text{vert}(\mathcal{X}))$, and
(ii) $\forall P \in \mathcal{X} \Longrightarrow \exists \mathcal{K} \subseteq \mathcal{T}$ such that $|\mathcal{K}| = P$.

Note that $\mathcal{T}$ may contain Steiner points. We define a *mesh* of $\mathcal{X}$ to be a subcomplex $\mathcal{K}$ of $\mathcal{T}$ such that $|\mathcal{K}| = |\mathcal{X}|$. According to our definitions, a triangulation of a set S of vertices triangulates the convex hull of S, while a mesh of S is just S itself. See Fig. 6 for examples. Our output object can be either a triangulation or a mesh of the input PLS.

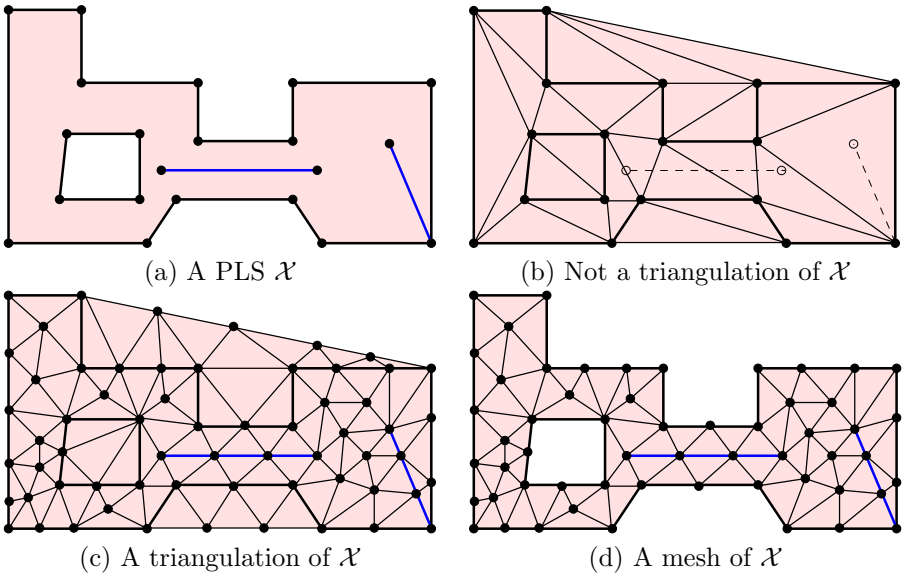
The Meshing Problem: Let $\mathcal{X}$ be a 3D PLS. Find a tetrahedral mesh $\mathcal{T}$ of $\mathcal{X}$ such that *(i)* the number of Steiner points in $\mathcal{T}$ is small, and *(ii)* the mesh quality of $\mathcal{T}$ is optimal.

Requirement *(i)* means that the number of Steiner points in $\mathcal{T}$ should not be arbitrarily large. One way to bound it, is to show that this number is within $O(n^p)$, where n is the input size (e.g., the number of vertices, segments, facets) of $\mathcal{X}$, and p is a constant.

The requirement *(ii)* has more meanings. First of all, the definition of mesh quality depends on applications. In the context of numerical simulations, the mesh quality is determined together by several measures on element shape, size, and orientation [31]. In 3D boundary conformity problem, there is no precise definition of its mesh quality. A necessary requirement is that the element edge length is not too short. This means that no Steiner point should be added arbitrarily close to an existing vertex. More formally, the shortest edge length in $\mathcal{T}$ should be a bounded mesh sizing function defined over $|\mathcal{X}|$.

3. CONSTRAINED DELAUNAY TETRAHEDRALIZATIONS

Let $S \subset \mathbb{R}^d$ be a finite set of vertices. A *Delaunay triangulation* for S is a simplicial complex whose union is the convex hull of S and every simplex is characterized by the *Delaunay*



(a) A PLS $\mathcal{X}$ (b) Not a triangulation of $\mathcal{X}$

(c) A triangulation of $\mathcal{X}$ (d) A mesh of $\mathcal{X}$

Figure 6. Triangulation and mesh.

criterion (also known as the "empty sphere" criterion) [32]: a simplex σ whose vertices are in S is *Delaunay* if there exists a circumscribed ball B_σ of σ such that B_σ contains no vertices of S in its interior. Delaunay triangulations have many optimal properties [33] which are useful in various applications. Efficient algorithms are proposed for computing Delaunay triangulations in two and three dimensions, see e.g., [34, 35, 36, 37, 38].

Let $\mathcal{X}$ be a PLS in $\mathbb{R}^d$. A *conforming Delaunay triangulation* of $\mathcal{X}$ is a triangulation of $\mathcal{X}$ such that every simplex of the triangulation is Delaunay. Note that Steiner points are usually needed in a conforming Delaunay triangulation of $\mathcal{X}$. Edelsbrunner *et al* [39] proved that one needs at most $O(n^3)$ Steiner points for obtaining a conforming Delaunay triangulation in $\mathbb{R}^2$. No upper bound is available yet in dimension higher than two.

Let $\mathcal{G}$ be a planar straight line graph. Any two edges of $\mathcal{G}$ are either disjoint or meet at a common endpoint (i.e., $\mathcal{G}$ is a 1-dimensional PLS). A *constrained Delaunay triangulation* of $\mathcal{G}$ is a triangulation $\mathcal{T}$ of $\mathcal{G}$ such that the circumscribed circle of any triangle $\tau \in \mathcal{T}$ contains no vertex of $\mathcal{T}$ which is visible from the interior of τ, see Fig. 3. This definition is independently developed by Lee and Lin [18] and Chew [19]. Note that Steiner points are not needed in this definition. This concept can be generalized into d dimensions for $d \geq 3$. While it is necessary to take Steiner points into account.

A crucial concept is the *visibility* of points in $\mathbb{R}^d$. The basic idea is: every polyhedron $P \in \mathcal{X}$ may block the visibility of points which are not in P, while P does not block the visibility for its own points. We say that two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are *invisible* to each other if there exists a polyhedron $P \in \mathcal{X}$ such that the interior of the line segment $\mathbf{xy}$ intersects P at a single point. Otherwise $\mathbf{x}$ and $\mathbf{y}$ are *visible* to each other. See Fig. 7 left for examples.

The next definition, referred as the *constrained Delaunay criterion*, relaxes the Delaunay criterion by using the visibility of points. Let S be a finite set of points and $\mathcal{X}$ be a PLS in $\mathbb{R}^d$

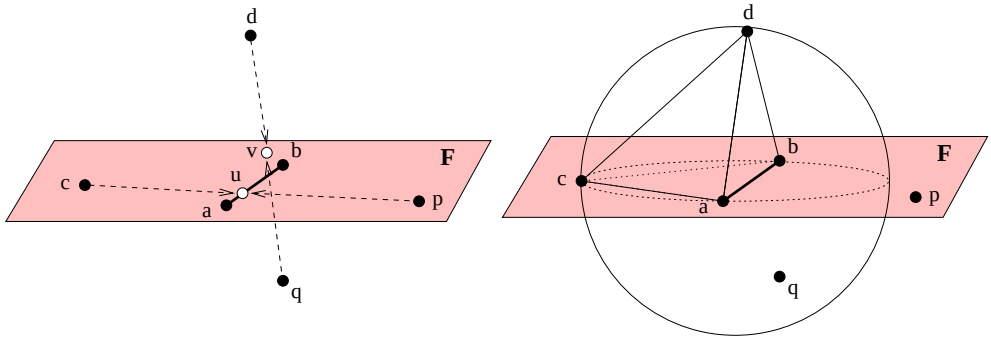


Figure 7. Visibility and constrained Delaunay criterion. The shaded region is a facet F of a PLS $\mathcal{X}$ in $\mathbb{R}^3$, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{p} \in F$. $\mathbf{ab}$ is a segment of $\mathcal{X}$. Left: $\mathbf{d}$ and $\mathbf{q}$ are invisible to each other since $\mathbf{dq} \cap F = \mathbf{v}$. $\mathbf{c}$ and $\mathbf{p}$ are invisible to each other since $\mathbf{cp} \cap \mathbf{ab} = \mathbf{u}$. $\mathbf{u}$ sees both $\mathbf{c}$ and $\mathbf{p}$. Right: A circumscribed ball of the tetrahedron $\mathbf{abcd}$ contains $\mathbf{q}$. $\mathbf{abcd}$ is constrained Delaunay since $\mathbf{q}$ is not visible from its interior. The triangle $\mathbf{abc} \subset F$ is constrained Delaunay since $\mathbf{p}$ is outside its diametric ball.

with $\text{vert}(\mathcal{X}) \subseteq S$. A simplex σ whose vertices are in S is *constrained Delaunay* if it is in one of the two cases:

(i) There is a circumscribed ball B_σ of σ which is empty.
(ii) Let F be the lowest-dimensional polyhedron in $\mathcal{X}$ that contains σ. Let $K = S \cap \text{aff}(F)$, then there is a circumscribed ball B_σ of σ that no vertex of K contained in the interior of B_σ is visible from any point in the interior of σ.

Case *(i)* means that every Delaunay simplex is constrained Delaunay. In *(ii)*, K is the subset of S in the affine hull generated by F. If a simplex $\sigma \subset F$ is constrained Delaunay or not, only depends on vertices of K. See Fig. 7 right for an example.

A *constrained Delaunay triangulation* (abbreviated as CDT) of $\mathcal{X}$ is defined as a triangulation $\mathcal{T}$ of $\mathcal{X}$ such that every simplex of $\mathcal{T}$ is constrained Delaunay. A three-dimensional CDT is also called a *constrained Delaunay tetrahedralization*.

By this definition, a CDT of $\mathcal{X}$ may contain Steiner points, i.e., the points in $S \setminus \text{vert}(\mathcal{X})$. It is called a *pure* CDT if it does not contain Steiner points. A 2-dimensional pure CDT is the same as the one defined by Lee and Lin [18] and Chew [40]. Shewchuk's definition of a CDT [41] is also a pure CDT.

In the following, we introduce some basic properties of the CDTs we've just defined. These properties show that a CDT of a PLS $\mathcal{X}$ is very close to a conforming Delaunay triangulation of $\mathcal{X}$. The proof are omitted, they are found in [26].

Delaunay triangulations can be checked locally. The following theorem shows that a CDT can be also checked locally.

Let $\mathcal{X}$ be a 3-dimensional PLS. Let $\mathcal{T}$ be any tetrahedralization of $\mathcal{X}$. A triangle σ of $\mathcal{T}$ is *locally Delaunay* if either *(i)* σ is on the boundary of the convex hull, or *(ii)* $\sigma \subset |\partial\mathcal{X}|$, or *(iii)* the opposite vertex of τ is not in the interior of B_ν of ν, where $\tau, \nu \in \mathcal{T}$ are the unique tetrahedra such that $\sigma = \tau \cap \nu$. Note that *(ii)* implies that a triangle which is contained in the boundary of $|\mathcal{X}|$ is automatically locally Delaunay.

Theorem 1 (Constrained Delaunay Lemma [26]) *If every triangles of $\mathcal{T}$ is locally Delaunay, then $\mathcal{T}$ is a CDT of $\mathcal{X}$.* ■

The key issue in the proof of the above theorem is to use the "Acyclic Theorem" of Edelsbrunner [42] which says that from any fixed viewpoint, a sequence of Delaunay simplices formed by the in_fromt/behind relation do not form a cycle. For each tetrahedra $\tau \in \mathcal{T}$ and a vertex $\mathbf{p} \in \mathcal{T}$ which is visible from the interior of τ, one can always form a finite sequence of tetrahedra $\{\tau = \tau_0, \tau_1, ..., \tau_k\}$ in $\mathcal{T}$ which all intersect a line segment $\mathbf{xp}$, where $\mathbf{x}$ is an interior point of τ. Then $\mathbf{p}$ must not lie in the interior of the circumscribed ball of τ by the Theorem.

If a point set S in $\mathbb{R}^3$ is in general position, i.e., no 5 points of S share a common sphere, then the Delaunay tetrahedralization of S is unique. By the above theorem, it is easy to show that this property holds for CDT as well.

Corollary 2. *Let $\mathcal{T}$ be a CDT of $\mathcal{X}$. If vert($\mathcal{T}$) is in general position, then $\mathcal{T}$ is the unique CDT of $\mathcal{X}$ with respect to vert($\mathcal{T}$).* ■

Let $\mathcal{X}$ be a 3-dimensional PLS. The i-skeleton $\mathcal{X}^{(i)}$ of $\mathcal{X}$ is an i-dimensional PLS, where $0 \leq i \leq 2$. It is useful to know the properties of a CDT of $\mathcal{X}^{(i)}$.

Theorem 3 ([26]) *Let $\mathcal{X}$ be a 3-dimensional PLS.*
(i) *A CDT of $\mathcal{X}^{(2)}$ is a CDT of $\mathcal{X}$.*
(ii) *A CDT of $\mathcal{X}^{(i)}$ is a conforming Delaunay tetrahedralization of $\mathcal{X}^{(i)}$, where $i = 0, 1$.* ■

Note that in *(ii)*, $\mathcal{X}^{(0)}$ (which is a set of points) and $\mathcal{X}^{(1)}$ (which is a set of segments and points) are 0-, and 1-dimensional PLSs embedded in 3-dimensional space, respectively. The elements in $\mathcal{X}^{(0)}$ and $\mathcal{X}^{(1)}$ do not block the visibility in a set of tetrahedra. Hence every tetrahedron in a CDT of $\mathcal{X}^{(0)}$ or $\mathcal{X}^{(1)}$ must be a Delaunay tetrahedron.

4. THE CDT ALGORITHM

Let $\mathcal{X}$ be a three-dimensional PLS, i.e., $\mathcal{X}$ is a collection of polyhedra of dimensions up to 3. The *boundary set* of $\mathcal{X}$ is its 2-skeleton, $\mathcal{X}^{(2)}$, which is the set of all vertices, segments, and facets of $\mathcal{X}$. The algorithm to construct a constrained Delaunay tetrahedralization $\mathcal{T}$ of $\mathcal{X}$ works in the following steps:

1. Initialize a CDT $\mathcal{D}_0$ of $\mathcal{X}^{(0)}$.
2. Let $\mathcal{D}_1 = \mathcal{D}_0$. Recover segments of $\mathcal{X}$ in $\mathcal{D}_1$ such that $\mathcal{D}_1$ is a CDT of $\mathcal{X}^{(1)}$.
3. Let $\mathcal{D}_2 = \mathcal{D}_1$. Recover facets of $\mathcal{X}$ in $\mathcal{D}_2$ such that $\mathcal{D}_2$ is a CDT of $\mathcal{X}^{(2)}$.

This algorithm proceeds in the increasing order of the dimensions of the skeletons. It initializes a CDT of $\mathcal{X}^{(0)}$ (which is a Delaunay tetrahedralization of vert($\mathcal{X}$)). This step can be completed efficiently by any of the Delaunay triangulation algorithms mentioned before.

The next two steps are crucial. They perform segment recovery and facet recovery by incrementally constructing a CDT $\mathcal{D}_i$ of $\mathcal{X}^{(i)}$, where $i = 1, 2$. We will discuss the details

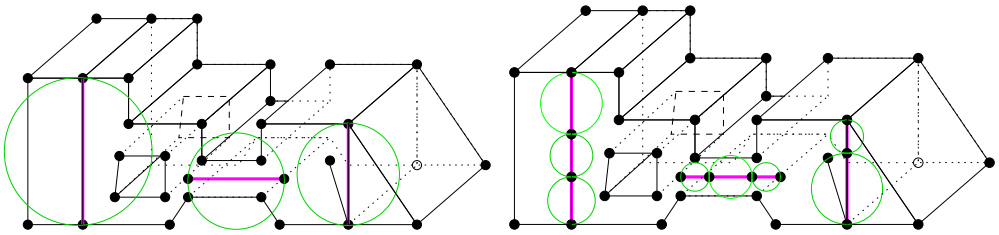


Figure 8. A condition (Theorem 5) guarantees the existence of a CDT without Steiner points. The left PLS does not satisfy the condition. The right PLS whose all edges are Delaunay satisfies the condition if the vertex set of the PLS is in general position.

in the next two sections, respectively. On finish of these two steps, By Theorem 3, $\mathcal{D}_2$ is a CDT of $\mathcal{X}$.

In this algorithm, we only add Steiner points in step *2* (segment recovery). Step *(3)* is done without adding Steiner points. In order to prove this, the following assumption is needed.

Assumption 4. *Assume the vertex set of the CDT $\mathcal{D}_1$ of $\mathcal{X}^{(1)}$ is in general position, i.e., no five vertices of vert($\mathcal{D}_1$) share a common sphere.*

Although this assumption is very strong, it can be satisfied easily by using techniques like symbolic perturbations [43, 20, 44]. Hence, theoretically, there is no need to actually perturb the vertices. We will discuss this issue in the implementation of this algorithm.

Shewchuk [28] showed that if every segment of $\mathcal{X}$ is Delaunay with respect to the vertex set of $\mathcal{X}$, then it is possible to recovery facets of $\mathcal{X}$ without using Steiner points. The following theorem follows directly from Shewchuk's result [28] and Corollary 2.

Theorem 5. *If the vertex set of $\mathcal{D}_1$ is in general position and $\mathcal{D}_1$ contains all segments of $\mathcal{X}$, then $\mathcal{X}$ has a unique CDT with no Steiner point.* ■

The above theorem states a slightly stronger condition than Shewchuk's condition [28] (see Fig. 8). Note that the latter does not require the general position assumption to be hold for the whole point set of $\mathcal{D}_1$. The success of this CDT algorithm is guaranteed.

5. SEGMENT RECOVERY

The Delaunay tetrahedralization $\mathcal{D}_0$ of vert($\mathcal{X}$) may not contain all segments of $\mathcal{X}$. This section presents a segment recovery algorithm for recovering missing segments of $\mathcal{X}$. The inputs are $\mathcal{X}^{(1)}$ and $\mathcal{D}_0$. The output of this algorithm is a CDT $\mathcal{D}_1$ of $\mathcal{X}^{(1)}$. Hence every segment of $\mathcal{X}$ is a union of edges of $\mathcal{D}_1$. For the mesh quality requirement, it is desired that no unnecessarily short edge is introduced in $\mathcal{D}_1$.

5.1. Segment Splitting Rules

Vertices are classified into two types. A vertex of $\mathcal{X}$ is *acute* if at least two segments of $\mathcal{X}$ incident at it form an angle less than 60^o, otherwise it is *non-acute*. A Steiner point is always

non-acute. *Remark*: The choice of 60^o is due to the fact that it is a safe value for the termination of Ruppert's Delaunay refinement algorithm [45].

A segment of $\mathcal{X}$ is transformed into two *subsegments* of $\mathcal{X}$ by inserting a Steiner point in it. Later on, unless it is explicitly mentioned, the term "segment" means either a segment or a subsegment. We distinguish two types of segments in $\mathcal{X}$: a segment is *type-0* if both endpoints of it are either non-acute or acute, it is *type-1* if only one of its endpoints is acute.

The *diametric ball* of a segment is by definition the smallest circumscribed ball of it. A vertex is said to *encroach upon* a segment if it lies inside the diametric ball of that segment. We have the following fact due to the Delaunay property of $\mathcal{D}_1$.

Fact 6. *If a segment of $\mathcal{X}$ is missing in $\mathcal{D}_1$ and vert($\mathcal{D}_1$) is in general position, then it must be encroached by at least one vertex of $\mathcal{D}_1$.*

The *local feature size* [45] lfs($\mathbf{v}$) of any point $\mathbf{v} \in |\mathcal{X}|$ is the radius of the smallest ball centered at $\mathbf{v}$ that intersects two disjoint elements of $\mathcal{X}$. The lfs() defines a continuous map that maps every point in $|\mathcal{X}|$ into a positive value which suggests how large the ball of the empty space around this point can be. This function only depends on the set $\mathcal{X}$ and does not change as new points are inserted.

Let $\mathbf{e}_i\mathbf{e}_j$ be a segment in $\mathcal{X}$ with endpoints $\mathbf{e}_i$ and $\mathbf{e}_j$. If $\mathbf{e}_i\mathbf{e}_j$ is missing in $\mathcal{D}_1$, it will be split by a Steiner point $\mathbf{v}$ in the interior of it. A *reference point* $\mathbf{p}$ of $\mathbf{v}$, which is responsible for the insertion of $\mathbf{v}$, is chosen randomly from the set of encroaching points of $\mathbf{e}_i\mathbf{e}_j$.

Let $\Sigma(\mathbf{c}, r)$ be a sphere with center $\mathbf{c}$ and radius r, and let $\|\cdot\|$ be the Euclidean distance function. Let $c > 0$ be a parameter given on the input. The choice of $\mathbf{v}$ is then governed by three rules given below:

1. If $\mathbf{e}_i\mathbf{e}_j$ is type-0 (Fig. 9 left), then $\mathbf{v} = \mathbf{e}_i\mathbf{e}_j \cap \Sigma(\mathbf{c}, r)$, where
   if $\|\mathbf{e}_i - \mathbf{p}\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$ **then**
   $\mathbf{c} = \mathbf{e}_i, r = \|\mathbf{e}_i - \mathbf{p}\|$;
   else if $\|\mathbf{e}_j - \mathbf{p}\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$ **then**
   $\mathbf{c} = \mathbf{e}_j, r = \|\mathbf{e}_j - \mathbf{p}\|$;
   else
   $\mathbf{c} = \mathbf{e}_i, r = \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_j\|$;
   end.
2. If $\mathbf{e}_i\mathbf{e}_j$ is type-1 and $\mathbf{e}_i$ is acute (Fig. 9 middle), then $\mathbf{v} = \mathbf{e}_i\mathbf{e}_j \cap \Sigma(\mathbf{c}, r)$, where $\mathbf{c} = \mathbf{e}_i$ and $r = \|\mathbf{e}_i - \mathbf{p}\|$. However, **if** $\|\mathbf{v} - \mathbf{e}_j\| < \|\mathbf{v} - \mathbf{p}\|$ and $r > \frac{1}{c}$lfs($\mathbf{e}_i$), **then** reject $\mathbf{v}$ and use Rule 3; **end**.
3. (Continued from Rule 2) Let $\mathbf{v}'$ be the rejected vertex by Rule 2 (Fig. 9 right), then $\mathbf{v} = \mathbf{e}_i\mathbf{e}_j \cap \Sigma(\mathbf{c}, r)$, where $\mathbf{c} = \mathbf{e}_i$, and
   if $\|\mathbf{p} - \mathbf{v}'\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$ **then**
   $r = \|\mathbf{e}_i - \mathbf{v}'\| - \|\mathbf{p} - \mathbf{v}'\|$;
   else
   $r = \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$;
   end.

The purpose of these rules is to avoid creating unnecessarily short edges. Note that Rule-1 never creates an edge shorter than the distance $\|\mathbf{e}_i - \mathbf{p}\|$. Rule-2 ensures that the edge $\mathbf{e}_i\mathbf{v}$ at the acute vertex $\mathbf{e}_i$ is at least $\|\mathbf{e}_i - \mathbf{p}\|$ as well. However, the edge $\mathbf{v}\mathbf{e}_j$ (not at $\mathbf{e}_i$) may be

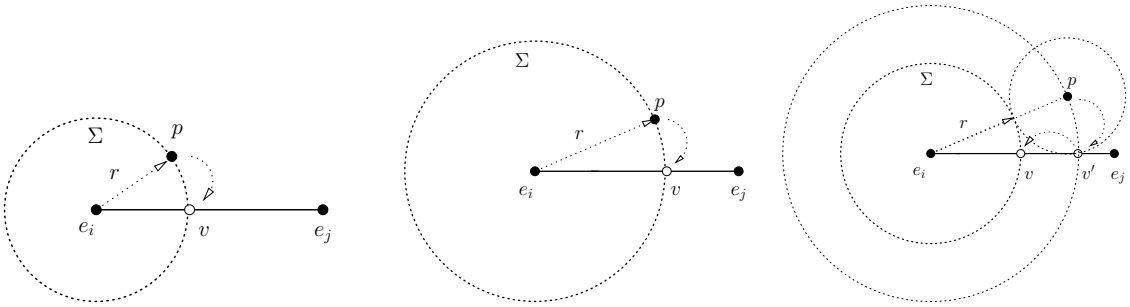


Figure 9. Segment splitting rules.

shorter than it. The apply of Rule-3 ensures that the edge $\mathbf{ve}_j$ will not be too short. Note that Rule-3 is only applied if the edge at acute vertex $\mathbf{e}_i$ is not less than $\frac{1}{c}\text{lfs}(\mathbf{e}_i)$. The condition $c > 0$ guarantees the termination of this algorithm, i.e., no edge less than half of this distance will be created.

For several segments sharing an acute vertex, by repeatedly using Rule 2, a protecting ball is automatically created which ensures: no other vertex can be inserted inside the ball. The effect is shown in Fig. 11 right. Note that the protecting ball is not necessarily completely created, i.e., only the missing segments will be split and protected. Existing segments remain untouched. This feature reduces the insertion of unnecessary Steiner points.

5.2. The Algorithm

The SegmentRecovery algorithm is described in Fig. 10. The inputs are a three-dimensional PLS $\mathcal{X}$ and a Delaunay tetrahedralization $\mathcal{D}_0$ of vert($\mathcal{X}$). The algorithm initializes $\mathcal{D}_1$ to be $\mathcal{D}_0$, and a set $\mathcal{S}$ of all segments of $\mathcal{X}$. Then it runs into a loop until $\mathcal{S}$ is empty.

At each time, a randomly selected segment $\mathbf{e}_i\mathbf{e}_j$ is removed from $\mathcal{S}$. If it is missing in $\mathcal{D}_1$, a Steiner point $\mathbf{v}$ is generated by one of the three segment splitting rules. $\mathbf{v}$ splits $\mathbf{e}_i\mathbf{e}_j$ into two subsegments $\mathbf{e}_i\mathbf{v}$ and $\mathbf{e}_j\mathbf{v}$, they are added into $\mathcal{S}$. Moreover, the insertion of $\mathbf{v}$ may cause other existing segments (subsegments) of $\mathcal{X}$ missing in $\mathcal{D}_1$, they are added into $\mathcal{S}$ as well. Here B_σ means the diametric circumscribed ball of a segment $\sigma \in \mathcal{X}^{(1)}$. $\mathcal{D}_1$ is updated to a Delaunay tetrahedralization of the vertex set including $\mathbf{v}$.

5.3. Comparison with another Algorithm

Shewchuk also proposed a segment recovery algorithm [20] for the same purpose. This algorithm proceeds in two steps. The first step uses protecting spheres centered at acute vertices of $\mathcal{X}$. New points are placed at where the segments and spheres intersect (see Fig. 11). The radii of these spheres are calculated in advance such that the subsegments inside the spheres are guaranteed to be Delaunay. The second step simply recovers other (sub)segments which are not Delaunay by recursive bisections. Fig 11 shows the results of the two algorithms on a 2D input, respectively. Comparing the two results, our algorithm adds less Steiner points and the created subsegments are longer.

Algorithm SegmentRecovery ($\mathcal{X}$, $\mathcal{D}_0$, c)
// $\mathcal{X}$ is a three-dimensional PLS; $\mathcal{D}_0$ is the DT of vert($\mathcal{X}$). $c > 0$.
1. $\mathcal{D}_1 = \mathcal{D}_0$;
2. Calculate the local feature sizes for all acute vertices;
3. Initialize a set $\mathcal{S}$ of all segments of $\mathcal{X}$;
4. **while** $\mathcal{S} \neq \emptyset$ **do**
5. get a segment $\mathbf{e}_i\mathbf{e}_j \in \mathcal{S}$; $\mathcal{S} = \mathcal{S} \setminus \{\mathbf{e}_i\mathbf{e}_j\}$;
6. **if** $\mathbf{e}_i\mathbf{e}_j$ is missing in $\mathcal{D}_1$, **then**
7. find a Steiner point $\mathbf{v} \in \text{int}(\mathbf{e}_i\mathbf{e}_j)$ by Rule-i, $i \in \{1, 2, 3\}$;
8. $\mathcal{S} = \mathcal{S} \cup \{\mathbf{e}_i\mathbf{v}, \mathbf{e}_j\mathbf{v}\}$;
9. $\mathcal{S} = \mathcal{S} \cup \{\sigma \in \mathcal{X}^{(1)}, \sigma \in \mathcal{D}_1 \,\big|\, \mathbf{v} \in \text{int}(B_\sigma)\}$;
10. update $\mathcal{D}_1$ to be the DT of vert($\mathcal{D}_1$) $\cup \{\mathbf{v}\}$;
11. update $\mathcal{X}$ to be the PLS including $\mathbf{v}$;
12. **endif**
13. **endwhile**
14. **return** $\mathcal{D}_1$;

Figure 10. The segment recovery algorithm.

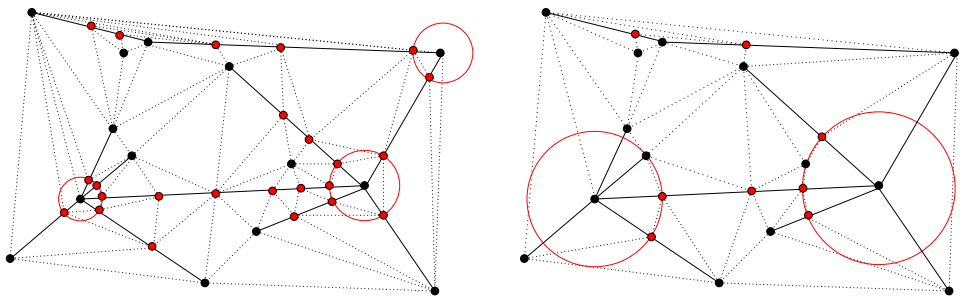


Figure 11. Comparison with Shewchuk's algorithm [20] (in 2D). The protecting balls around acute vertices are shown in circles. Left: a result of Shewchuk's algorithm, 24 Steiner points. Right: a result of our algorithm, 8 Steiner points.

6. FACET RECOVERY

The input of the facet recovery algorithm is a CDT $\mathcal{D}_1$ of $\mathcal{X}^{(1)}$. Some facets of $\mathcal{X}$ may not be represented by $\mathcal{D}_1$. This step does not need Steiner points.

6.1. The Algorithm

At initialization, let $\mathcal{D}_2 = \mathcal{D}_1$. Each facet $F \in \mathcal{X}$ together with the Steiner points inserted on F is first triangulated into a two-dimensional CDT $\mathcal{T}_F$. We call triangles of $\mathcal{T}_F$ *subfaces* to distinguish other faces of $\mathcal{D}_2$. Some subfaces may be missing in $\mathcal{D}_2$. Add all missing subfaces of $\mathcal{T}_F$ into a set $\mathcal{S}$. The facet recovery algorithm incrementally recovers missing subfaces of $\mathcal{T}_F$ and updates $\mathcal{D}_2$, it stops when $\mathcal{S}$ is empty.

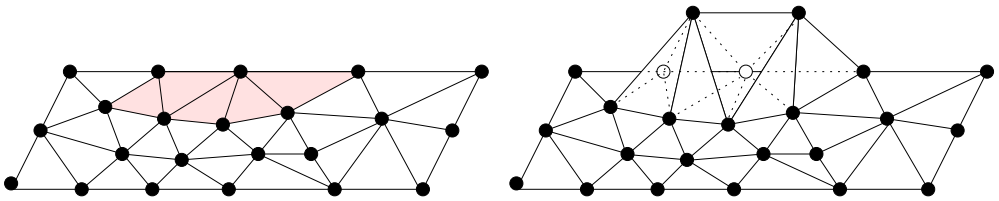


Figure 12. Left: The shaded area highlights a missing region Ω. Right: One of the cavities resulting from a missing region is illustrated.

At each iteration i, several missing subfaces of $\mathcal{T}_F$ are recovered together. We define a *missing region* Ω to be a set of subfaces of $\mathcal{T}_F$ such that

(i) the edges on the boundary of Ω are edges of $\mathcal{D}_2$, and

(ii) the edges in the interior of Ω are missing in $\mathcal{D}_2$.

Hence Ω is a connected set of missing subfaces. It may not be simply connected. Each missing subface belongs to a missing region. A facet can have more than one missing region.

When a missing region Ω is found, one can derive a cavity in $|\mathcal{D}_2|$ by removing all tetrahedra whose interiors intersect with Ω. This cavity can be further subdivided into two cavities by inserting the subfaces of Ω in it, see Fig. 12 right. Each cavity is a three-dimensional polyhedron C whose facets are triangles. The boundary $\mathrm{bd}(C)$ of C is the union of triangular faces which are either subfaces of Ω or faces in $\mathcal{D}_2$.

The next step is to tetrahedralize each cavity C without using Steiner points. The TetrahedralizeCavity algorithm (given in Fig. 13) has two phases, *(1)* cavity verification (lines $2-13$), and *(2)* cavity tetrahedralization (lines $15-20$).

In phase *(1)*, each face $\sigma \subset \mathrm{bd}(C)$ is verified. If σ is not Delaunay with respect to the vertex set of C, then C is enlarged by including the tetrahedron $\tau \in \mathcal{D}_2$, $\sigma < \tau$ and τ is removed from $\mathcal{D}_2$ (line 7). $\mathrm{bd}(C)$ is also changed, hence the three faces of τ are added into Q_C for later processing (line 8). See Fig. 14 for an example. If σ is a subface of $\mathcal{X}$, the enlargement of C will cause σ missing from $\mathcal{D}_2$. For this reason, we have to queue σ in Q (lines $9-11$) so it will be recovered later.

Phase *(2)* first constructs the Delaunay tetrahedralization $\mathcal{D}_C$ of $\mathrm{vert}(C)$ (line 16). By assumption 4, $\mathrm{vert}(C)$ is in general position, hence $\mathcal{D}_C$ will include all faces contained in $\mathrm{bd}(C)$. In other words, the interior of C is filled by a subset of tetrahedra of $\mathcal{D}_C$. The next step is to remove those tetrahedra which are not in $\mathrm{int}(C)$ from $\mathcal{D}_C$ (lines $16-20$). Fig. 15 illustrates the idea of phase *(2)* in two dimensions.

The FacetRecovery algorithm is given in Fig. 16. It recovers facets of $\mathcal{X}$ incrementally. Let $\mathcal{T}_F$ be a CDT of a facet $F \in \mathcal{X}$. It first initializes a queue Q of all subfaces of $\mathcal{T}_F$. Then it runs into a loop until Q is empty. Once a subface σ is found missing in $\mathcal{D}_2$, a missing region Ω containing σ is formed (line 6). All tetrahedra intersecting Ω are removed from $\mathcal{D}_2$. Two cavities C_1 and C_2 separated by Ω are formed in the interior of $|\mathcal{D}_2|$ (line 7). Then C_1 and C_2 are partitioned into two sets ($\mathcal{D}_{C_1}$ and $\mathcal{D}_{C_2}$) of tetrahedra by the subroutine TetrahedralizeCavity (lines $8-10$), respectively. $\mathcal{D}_2$ is then updated to respect Ω with the new partitions of C_1 and C_2 (lines 12). Fig. 15 illustrates the idea of this algorithm in two dimensions.

TetrahedralizeCavity ($\mathcal{D}_2$, C, Q)
// $\mathcal{D}_2$ is a tetrahedralization; C is a cavity; Q is a queue.
1. // Phase *(1)*: cavity verification.
2. initialize a queue Q_C of all faces contained in bd(C);
3. **while** $Q_C \neq \emptyset$ **do**
4. get a face $\sigma \in Q_C$; $Q_C = Q_C \setminus \{\sigma\}$;
5. **if** $\sigma \subset$ bd(C) and σ is non-Delaunay wrt. vert(C), **then**
6. get the tetrahedron $\tau \in \mathcal{D}_2$ such that $\sigma < \tau$;
7. $C = C \cup \tau$, $\mathcal{D}_2 = \mathcal{D}_2 \setminus \{\tau\}$;
8. $Q_C = Q_C \cup \{\nu < \tau \,|\, \nu \neq \sigma\}$;
9. **if** σ is a subface of $\mathcal{X}$, **then**
10. $Q = Q \cup \{\sigma\}$;
11. **endif**
12. **endif**
13. **endwhile**
14. // Phase *(2)*: cavity tetrahedralization.
15. form the Delaunay tetrahedralization $\mathcal{D}_C$ of vert(C);
16. **for** each tetrahedron $\tau \in \mathcal{D}_C$, **do**
17. **if** $\tau \not\subseteq$ int(C), **then**
18. $\mathcal{D}_C = \mathcal{D}_C \setminus \{\tau\}$;
19. **endif**
20. **endfor**
21. **return** $\mathcal{D}_2 = \mathcal{D}_2 \cup \mathcal{D}_C$;

Figure 13. The cavity re-tetrahedralization algorithm.

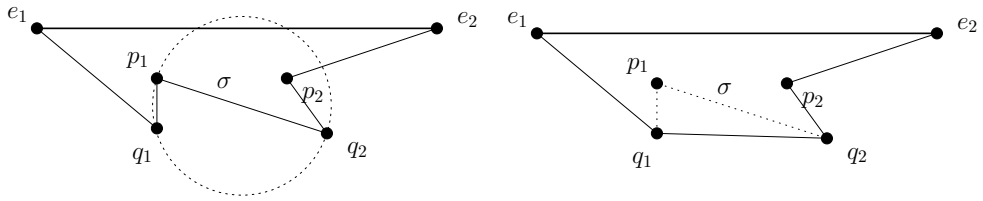


Figure 14. The verification of the cavity (illustrated in 2D). Left: $\mathbf{e_1e_2}$ is the segment being recovered. The cavity C formed from $\mathbf{e_1e_2}$ is a non-convex polygon. The edge $\mathbf{p_1q_2} \subset$ bd(C) is not Delaunay with respect to the vertex set of C. Right: C is enlarged by adding the triangle $\mathbf{q_1q_2p_1}$ into C. As a result, the edges $\mathbf{p_1p_2}$ and $\mathbf{p_1q_1}$ are not on bd(C) anymore, and the edge $\mathbf{q_1q_2}$ is included in bd(C).

6.2. Proof of Termination

The Assumption 4, i.e., the vertex set of $\mathcal{D}_1$ is in general position, is important. First, we need to show that the enlargement of the cavity C will terminate, i.e., the **do-while** loop (lines $3-14$) of the TetrahedralizeCavity algorithm will not run forever.

First note that the TetrahedralizeCavity algorithm terminates. The phase *(1)* (cavity verification) of the algorithm guarantees that the phase *(2)* can be correctly executed since every Delaunay simplex in vert(C) will appear in the Delaunay tetrahedralization of vert(C)

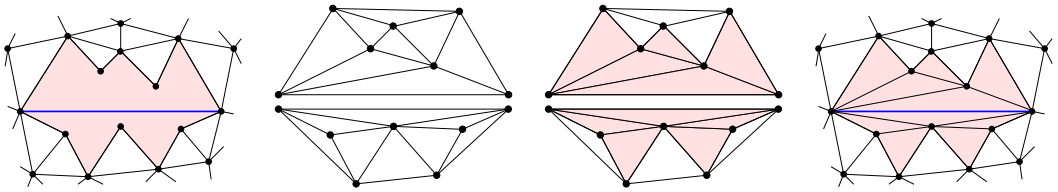


Figure 15. Cavity tetrahedralization (illustrated in 2D). (a) Two initial cavities separated by a constraining segment. (b) The two Delaunay triangulations are constructed at each side of the segment. (c) Classify triangles as "inside" or "outside". (d) Remove "outside" triangles.

FacetRecovery ($\mathcal{X}$, $\mathcal{D}_1$)
// $\mathcal{X}$ is a three-dimensional PLS; $\mathcal{D}_1$ is the CDT of $\mathcal{X}^{(1)}$.
1. $\mathcal{D}_2 = \mathcal{D}_1$;
2. Initialize a queue Q of all subfaces of $\mathcal{X}$;
3. **while** $Q \neq \emptyset$ **do**
4. get a subface $\sigma \in Q$; $Q = Q \setminus \{\sigma\}$;
5. **if** σ is missing in $\mathcal{D}_2$, **then**
6. form a missing region Ω containing σ;
7. form two cavities C_1, C_2 from Ω;
8. **for** each $C_i, i = \{1, 2\}$ **do**
9. TetrahedralizeCavity($\mathcal{D}_2$, C_i, Q);
10. **endfor**
11. **endif**
12. **endwhile**
13. **return** $\mathcal{D}_2$;

Figure 16. The facet recovery algorithm.

(providing the fact that vert(C) is in general position). Let $\mathcal{F} \neq \emptyset$ be the set of faces of $\mathcal{D}_2^{(2)}$ (the 2-skeleton of $\mathcal{D}_2$) such that no face of $\mathcal{F}$ is crossed by any subfaces of $\mathcal{X}$, for example, a convex hull face of $\mathcal{D}_2^{(2)}$ must be in $\mathcal{F}$. Clearly, any face $\sigma \in \mathcal{F}$ is Delaunay with respect to vert($\mathcal{D}_2$). It is also true that any face $\sigma \in \mathcal{F}$ is Delaunay with respect to vert(C). The set $\mathcal{F}$ limits the enlargement of C, i.e., C stops expanding at a $\sigma \in \mathcal{F}$.

There is a potential case of deadlock. In the TetrahedralizeCavity algorithm, after the recovery of a missing region Ω, some originally existing subfaces may be missing from $\mathcal{D}_2$, they are queued and will be recovered later (lines $9-11$). If the recovery of such subfaces (by the FacetRecovery algorithm) will cause Ω (or part of it) to be missing from $\mathcal{D}_2$ again, then a deadlock may appear. The next lemma shows that a deadlock will not happen.

Lemma 7. *Let Ω be a missing region, C is one of the cavities formed from Ω. Suppose there is a subface $\sigma \subset C$ (hence it is missing from $\mathcal{D}_2$ after the recovery of Ω). Let Ω_σ be the missing region containing σ formed immediately after the recovery of Ω, and let C_σ be any of the two cavities formed from Ω_σ in the* TetrahedralizeCavity *algorithm. Then $|C_\sigma| \subset |C|$.*

Proof By the definition of a missing region, Ω_σ is contained in C (since all edges on bd(C)

are edges of $\mathcal{D}_2$). It further implies that the initial cavity C_σ formed from Ω_σ is contained in C. Note that C_σ may be enlarged in the TetrahedralizeCavity algorithm.

Since any face $\nu \subset \text{bd}(C)$ is Delaunay with respect to vert(C), ν is also Delaunay with respect to the initial vert(C_σ). Hence C_σ could not include any tetrahedron outside C since any face $\nu \subset \text{bd}(C)$ limits the enlargement of C_σ. This implies that $|C_\sigma| \subset |C|$. ■

Note that there is no ordering on the subfaces in Q in the FacetRecovery algorithm. While Lemma 7 only guarantees no deadlock if the algorithm performs a specific recovery ordering on the subfaces in Q. We thus prove a "weak" termination of the algorithm which depends on such a specific ordering.

Theorem 8. *Suppose that the assumption 4 is satisfied. Moreover, the subfaces in Q are randomly ordered with one exception, that is, the subfaces queued in the* Tetrahedralizecavity *algorithm are always processed first. Then the* FacetRecovery *algorithm terminates.*

Proof Each missing region will be recovered. Lemma 7 and the specific ordering in Q together ensure that two missing regions can not form a deadlock. Hence Q will be empty and $\mathcal{D}_2$ contains all subfaces of $\mathcal{X}$. ■

In practice, we found that the termination of this algorithm does not depend on any ordering. For example, a random ordering in Q always terminated.

6.3. Correctness

In this section, we establish the correctness of the FacetRecovery algorithm, i.e., the resulting tetrahedralization $\mathcal{D}_2$ is a CDT of $\mathcal{X}$.

Theorem 9. *$\mathcal{D}_2$ is a CDT of $\mathcal{X}$.*

Proof The FacetRecovery algorithm is an iterative process. In each iteration, a missing region is recovered and $\mathcal{D}_2$ is updated. Let $\mathcal{D}_2^i$ denote the tetrahedralization after the recovery of i-th missing region, where $i = 0, 1, ..., m$. Hence initially, $\mathcal{D}_2^0 = \mathcal{D}_1$. Since the algorithm terminates, m is a finite number. We want to show that $\mathcal{D}_2^m$ is a CDT of $\mathcal{X}$.

We first show that $\mathcal{D}_2^1$ is a CDT. Tetrahedra of $\mathcal{D}_2^1$ which are outside the two cavities remain Delaunay. Let $\tau \in \mathcal{D}_2^1$ be a tetrahedron created inside a cavity C, and B_τ be its circumscribed ball. Let $\tau' \in \mathcal{D}_2^1$ be another tetrahedron sharing a face σ with τ. Let $\mathbf{v}$ be the vertex of τ' opposite to σ. We have the following cases:

(1) σ is a face inside C. Then $\tau' \subset C$, and B_τ must not contain $\mathbf{v}$ (guaranteed by the TetrahedralizeCavity algorithm).
(2) σ is a subface (hence $\sigma \subset \text{bd}(C)$). Then τ is constrained Delaunay even if B_τ contains $\mathbf{v}$, i.e, the inside of τ is not visible by $\mathbf{v}$. Otherwise, if $\mathbf{v}$ is visible by a point $\mathbf{x} \in \tau$, there exists a non-locally Delaunay face in $\mathcal{D}_2^0$ which intersects the line segment $\mathbf{xv}$, which means that $\mathcal{D}_2^0$ is not a Delaunay tetrahedralization - a contradiction;
(3) σ is not a subface and $\sigma \subset \text{bd}(C)$. Then $\tau' \nsubseteq C$, and B_τ must not contain $\mathbf{v}$. Otherwise, $\mathcal{D}_1$ is not a Delaunay tetrahedralization since the circumscribed ball of τ' is not empty, i.e., it contains the vertex of τ opposite to σ.

By induction, after iteration i, where $i > 1$, $\mathcal{D}_2^i$ is a CDT. Now we want to show that $\mathcal{D}_2^{i+1}$ is a CDT. Using the similar arguments as above, the only difference is in the case *(3)* which is given below:

(4) σ is not a subface and $\sigma \subset \mathrm{bd}(C)$. Then B_σ must not contain $\mathbf{v}$. Otherwise, $\mathcal{D}_2^i$ is not a CDT since the circumscribed ball of τ' contains the vertex of τ opposite to σ which is also visible from the interior of τ'.

Thus after all missing regions are recovered, $\mathcal{D}_2 = \mathcal{D}_2^m$ is a CDT of $\mathcal{X}$. ■

6.4. Running time estimation

In this section we consider the behavior of the FacetRecovery algorithm with respect to the number of vertices and facets of the input PLS.

Theorem 10. *Let $\mathcal{X}$ be a three-dimensional PLS which has v vertices. One missing facet can be recovered in expected time $O(v^2 + v \log v)$.*

Proof We prove this theorem by constructing a PLS which needs such a running time. Further it is shown that it is indeed the worst case.

The PLS $\mathcal{L}_{v,1}$ (1-layer) shown in Fig. 17 (a) has only 1 facet. It is a slightly perturbed square (to make it non-degenerate). $\mathcal{L}_{v,1}$ contains a set of v points (here $v = 100$) which are randomly distributed slightly above the facet. The diameter of the facet is much larger than that of the point set. There is another point in $\mathcal{L}_{v,1}$ which lies below the center of the facet. The Delaunay tetrahedralization of $\mathrm{vert}(\mathcal{L}_{v,1})$ is shown in (b). The facet of $\mathcal{L}_{v,1}$ is missing in it. The cavity formed from the missing facet has size $O(v)$. The CDT of $\mathcal{L}_{v,1}$ is shown in (c).

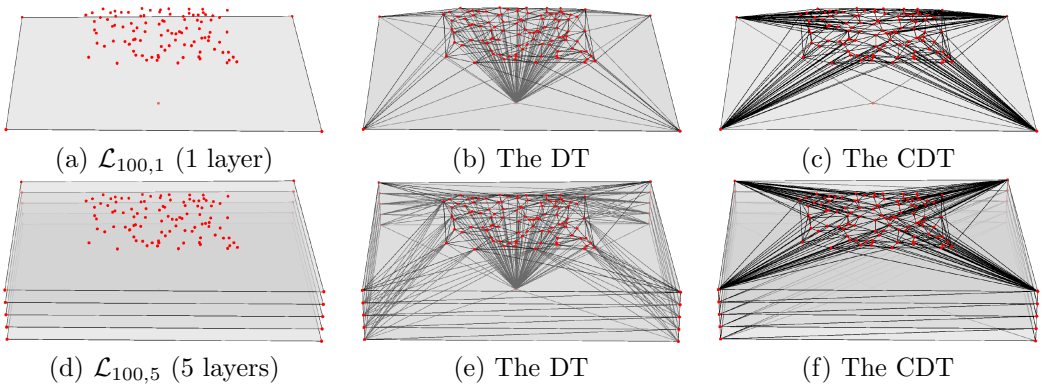
Note that the removal of outside tetrahedra (lines $2-6$ in TetrahedralizeCavity) takes linear time. The time for recovery of this facet is dominated by the time for constructing the Delaunay tetrahedralization of the set of v vertices. There exist point sets with linear size Delaunay tetrahedralizations that reach quadratic intermediate size with positive constant probability. By using the randomized incremental flip algorithm [37], the expected time for constructing the Delaunay tetrahedralization is at most $O(v^2 + v \log v)$. Note that the largest possible size of a cavity is v. These statements together prove the claim. ■

Next we show an example that the FacetRecovery algorithm runs in time $\Omega(fv^2 + fv \log v)$, where f are the number of facets.

Let $\mathcal{L}_{v,f}$ be the PLS extended from $\mathcal{L}_{v,1}$ by including a set of f parallel facets. Fig. 17 (d) shows an example for $f = 5$. Clearly, the Delaunay tetrahedralization of $\mathrm{vert}(\mathcal{L}_{v,f})$ will not contain the f facets, see (e). If the set of missing facets are recovered in an order which is from bottom to top, then the size of each cavity remains $O(v)$. This implies that the total time for recovery of the f facets is at least $O(fv^2 + fv \log v)$.

Remark: Note that the order of the recovery of the missing facets is important. For example, if we reverse the recovering order in the above proof, i.e., missing facets are recovered from top to bottom, only the top facet needs time $O(v^2 + v \log v)$, while the size of all other cavities is a constant (here it is 4).

The worst case complexity of this algorithm remains open. The main difficulty is that there are facets which may be recovered multiple times (due to the cavity enlargement procedure in



(a) $\mathcal{L}_{100,1}$ (1 layer) (b) The DT (c) The CDT

(d) $\mathcal{L}_{100,5}$ (5 layers) (e) The DT (f) The CDT

Figure 17. Examples (Layers). The PLS $\mathcal{L}_{100,1}$ shown in (a) has 1 facet (1 layer), 100 vertices lie on top of the facet, one vertex below the center of the facet. The facet is missing in the Delaunay tetrahedralization of vert($\mathcal{L}_{100,1}$) shown in (b). The CDT of $\mathcal{L}_{100,1}$ is shown in (c). The PLS $\mathcal{L}_{100,5}$ shown in (d) is extended from $\mathcal{L}_{100,1}$ by including 5 parallel facets. (e) and (f) respectively show the Delaunay tetrahedralization and the CDT of $\mathcal{L}_{100,5}$.

TetrahedralizeCavity). In our practice of this algorithm, it is very seldom that we need to recover a facet twice or more times.

7. BOUNDARY STEINER POINTS REMOVAL

In this section, we consider the related meshing problem: Given a surface triangular mesh $\mathcal{F}$ of a three-dimensional PLS $\mathcal{X}$, we want to find a tetrahedral mesh $\mathcal{T}$ of $\mathcal{X}$ such that $\mathcal{F}$ is a subcomplex of $\mathcal{T}$.

The three-dimensional CDT algorithm proposed in this paper will add Steiner points on elements of $\mathcal{F}$, and all of them are added on edges of $\mathcal{F}$. We then must remove these Steiner points by either suppressing them or by relocating them into the interior of $\mathcal{X}$. Of course the resulting mesh is not a CDT anymore. But this is not the question here.

7.1. Point Relocation

It is shown in [7] that any Steiner point inserted on an element (edge or face) of $\mathcal{F}$ is guaranteed to be removed from that element. The proposed approach is straightforward. Consider a Steiner point $\mathbf{v}$ inserted on a face $F \in \mathcal{F}$. Assume that F is an external face. One can form a "half-ball" $\mathcal{B}_v$ of $\mathbf{v}$ which consists of all tetrahedra in $\mathcal{T}$ having $\mathbf{v}$ as a vertex. The boundary $\partial\mathcal{B}_v$ are triangular faces either *(i)* coplanar with $\mathbf{v}$, or *(ii)* visible by $\mathbf{v}$ (since they form tetrahedra with v), see Fig. 18 left for an example. Hence it is always possible to relocate $\mathbf{v}$ inside $\mathcal{B}_v$ such that $\mathcal{B}_v$ becomes a "full-ball" of $\mathbf{v}$, see Fig. 18 middle. As a result, $\mathbf{v}$ has been removed from F. Steiner points inserted on edges of $\mathcal{F}$ can be removed by the same principle. The only difference is that an edge may be shared by arbitrary number of faces. Hence it may produce many half-balls. Within each one the Steiner point from the edge can be relocated.

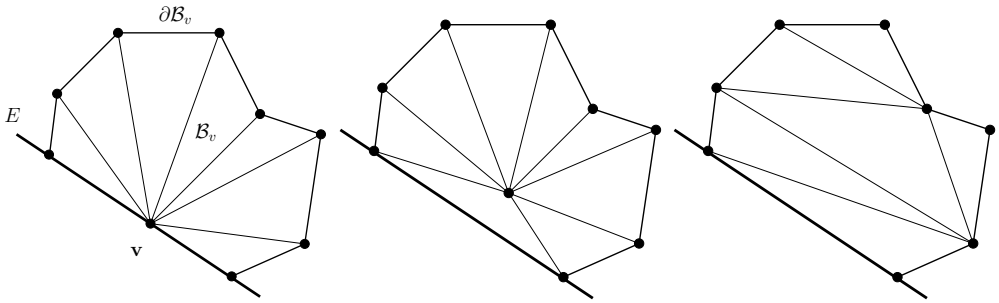


Figure 18. Boundary Steiner points removal (the 2D case). Left: **v** is a Steiner point on edge E. The half-ball $\mathcal{B}_v$ consists of all triangles having **v** as a vertex in one region. The bold lines are the boundary complex $\partial\mathcal{B}_v$. Middle: **v** is relocated into the interior of $\mathcal{B}_v$. Right: After **v** is deleted from $\mathcal{B}_v$.

7.2. Point Deletion

The relocated Steiner points may be completely removed from the mesh. The most common approach for this purpose is *edge contraction* – the Steiner points are removed by contracting edges to zero length. Although the operation is easy in principle, it is not trivial to select one edge to be contracted such that the resulting shapes of tetrahedra are optimal. This issue is addressed in [46] in which several criteria for contracting edges are proposed.

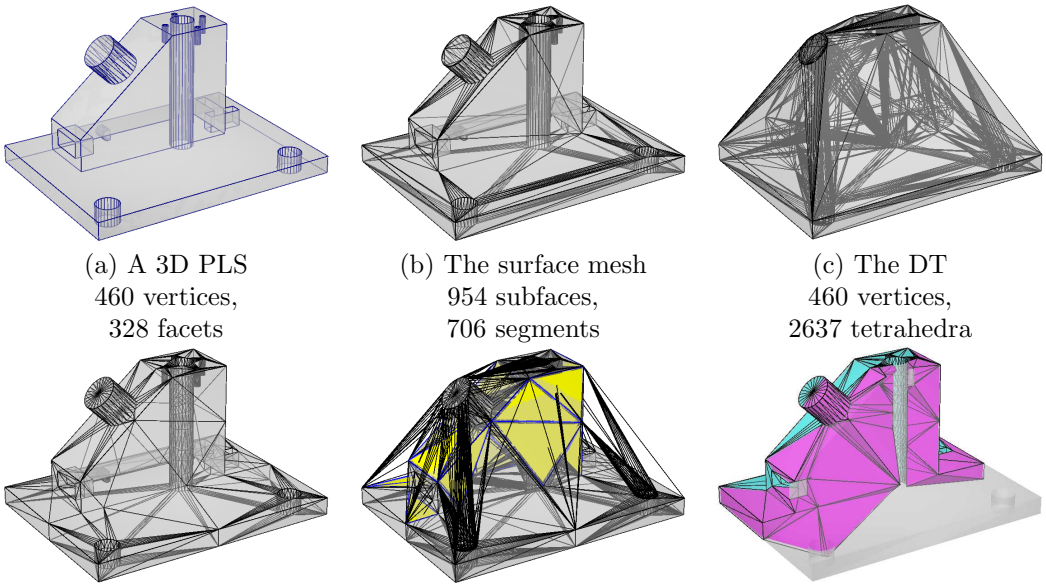
Another approach to directly delete **v** after forming the half-ball $\mathcal{B}_v$ is to re-tetrahedralize $\mathcal{B}_v$ such that the new tetrahedralization of $\mathcal{B}_v$ does not contain **v**. A *flip-based approach* works in the following steps,

1. Re-triangulate the boundary of $\mathcal{B}_v$ such that **v** is not on $\partial\mathcal{B}_v$ any more.
2. Form a Delaunay tetrahedralization $\mathcal{D}$ of the vertices of $\mathcal{B}_v$ (without **v**);
3. Recover the faces of $\partial\mathcal{B}_v$ in $\mathcal{D}$ by combination of edge/face flips;
4. If all faces of $\partial\mathcal{B}_v$ are recovered, remove tetrahedra outside $\mathcal{B}_v$ from $\mathcal{D}$; return $\mathcal{D}$; Otherwise, return $\emptyset$.

Although the above approach does not guarantee to work for all cases, it is shown in [47] that the flipping algorithm is effective in forming a tetrahedralization of $\mathcal{B}_v$. In Section 8, experiments on selected mesh examples are reported (see Table II).

7.3. Mesh Optimization

It is known that not all relocated points can be deleted, for example, when $\mathcal{F}$ is the surface mesh of a Schönhardt polyhedron. Moreover, some relocated points may be very close to the boundary faces (or edges). The resulting mesh must be optimized. Common mesh improvement techniques are local edge (or face) swapping, mesh smoothing, and even new point insertion. These techniques can be appropriately combined in optimizing some pre-defined mesh objective functions, see e.g. [48, 49].



(a) A 3D PLS
460 vertices,
328 facets

(b) The surface mesh
954 subfaces,
706 segments

(c) The DT
460 vertices,
2637 tetrahedra

(d) Segment recovery
213 break points
269 protect points

(e) Facet recovery
6446 subfaces
22 are missing

(f) A CDT
482 Steiner points
2738 tetrahedra

Figure 19. Example: `Cami1a`. The input PLS and the constructed CDT are shown in (a) and (f), respectively. Pictures from (b) to (e) show the intermediate status of the CDT algorithm.

8. EXAMPLES AND DISCUSSIONS

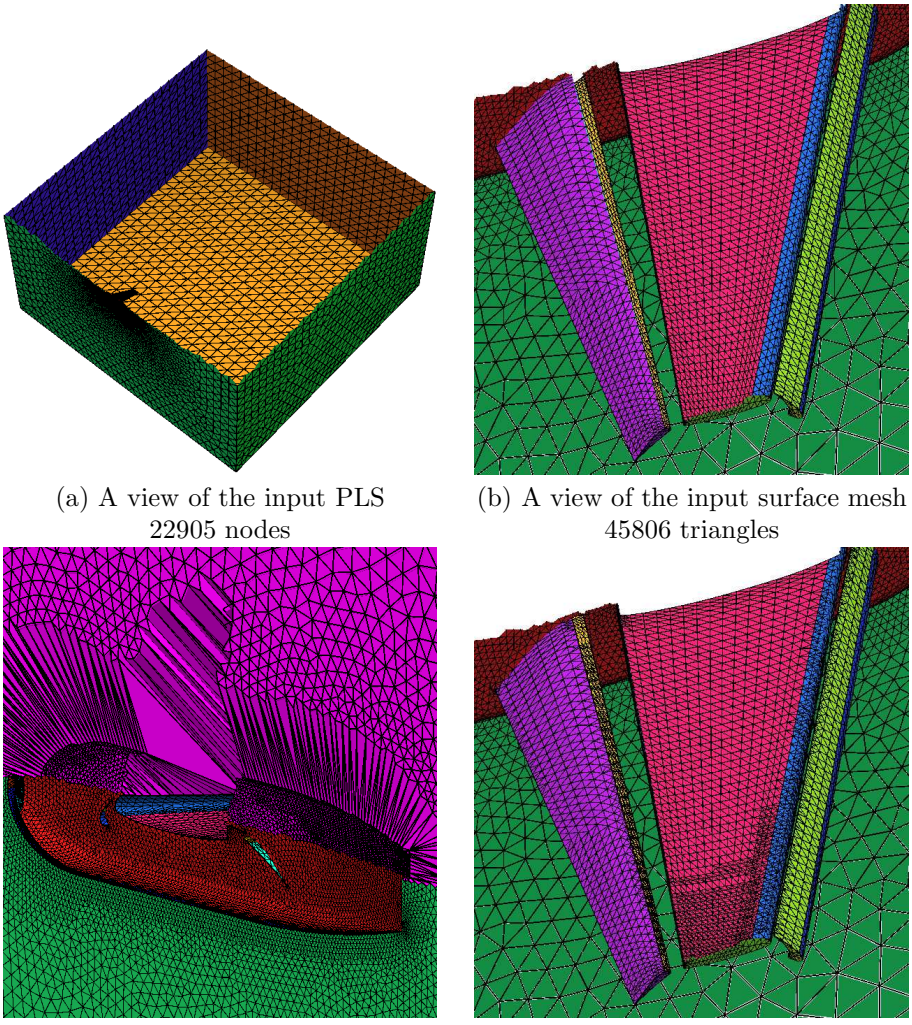
The CDT algorithm has been implemented in the program `TetGen` [50]. In this section, we provide several application examples to illustrate the practical behavior and the effectiveness of the CDT algorithm. Using the examples, we discuss the complexity of this algorithm and possible improvements.

The assumption 4 can be fulfilled by using techniques of symbolic perturbation [43, 20, 44]. In practice, it is helpful to actually add some Steiner points (so-called *break points*) to remove the *local degeneracies* [21] in the facets of the PLS. In the implementation of TetGen, both techniques are used. A symbolic perturbation approach [20] is used through out the CDT algorithm. In addition, a local degeneracy removal algorithm [21] is called before the segment recovery algorithm.

Fig. 19 illustrates an example of one run of the CDT algorithm on a mechanical part (`Cami1a`, available from [51]) with the intermediate status of the different steps. The input PLS shown in (a), it has 460 vertices, 706 segments, and 328 facets. The surface mesh is shown in (b). It is the input of the local degeneracy removal algorithm and contains 954 subfaces. (c) is the initial Delaunay tetrahedralization of the vertex set. The status after the SegmentRecovery algorithms is shown in (d). The number of break points and protect points are 213 and 269, respectively. (e) shows the initial status of the FacetRecovery algorithm, there are 6446 subfaces in which 22 are missing (highlighted). The resulting CDT is shown in (f). A vertical

cut is used to visualize the interior constrained Delaunay tetrahedra.

In the above example, the number of Steiner points is in the order of the input size. This is a typical case observed in practice. Using an appropriate point insertion order, the current one is random, it will be possible to reduce the number of Steiner points. The local degeneracy removal algorithm may be called only when it is necessary. It currently may add at most $O(n)$ Steiner points.



(a) A view of the input PLS
22905 nodes

(b) A view of the input surface mesh
45806 triangles

(c) A view of the output CDT
43044 nodes
134700 tetrahedra

(d) A view of the output surface mesh
2542 break points
17597 protect points

Figure 20. Example: `Wing-Iso`. A global and a local views of the input PLS are shown in (a) and (b), respectively. Two detailed views of the output CDT are shown in (c) and (d).

1		Cami1a	Heart	Fan	Crystal	Wing-Iso	IFP
2	Input nodes	460	3,588	6,516	11,706	22,905	57,270
3	Input segs	706	11,205	19,709	37,785	46,693	172,001
4	Input facets	884	7,620	13,180	26,399	45,806	114,680
5	Break points	213	0	102	8	2,542	0
6	Protect points	269	3,622	4,992	10,789	16,913	1,963
7	Delaunay tetra.	0.05	0.22	0.41	0.81	1.61	4.85
8	Surface mesh	0.02	0.03	0.10	0.15	0.44	0.38
9	L.d. removal	0.07	0.10	0.18	0.42	0.42	1.80
10	Seg. recovery	0.12	0.28	0.37	0.97	2.02	0.51
11	Facet recovery	0	0.06	0.07	0.64	0.13	0.31
12	Total time (sec.)	0.26	0.69	1.13	2.99	4.62	7.85

Table I. Runtime statistics of the CDT algorithm. Tested by `TetGen` (compiled by g++ with -O3 option) on a linux workstation (Intel(R) Xeon(R) CPU 2.40GHz). (In line 9, L.d. = Local degeneracy.)

The geometry of the next example shown in Fig. 20 is the wing of an airplane placed inside a large bounding box, see (a). The surface of the wing and the bounding box were triangulated by 22905 nodes and 45806 triangles. A detailed view of the surface triangulation of the wing is shown in (b). To generate the CDT from the surface mesh, `TetGen` added 19,455 Steiner points, 2542 are break points and 16,913 are protect points. A view of the inside of the CDT near the wing is shown in (c). In (d), the modified surface mesh of the CDT is shown.

We next report the detailed run times of the CDT algorithm on some selected examples in Table I. Most of the input PLSs (whose boundary are triangular surface meshes) are available from the repository of `3D Meshes Research Database` maintained by INRIA's GAMMA project [51]. Two of the generated CDTs are shown in Fig. 21. Table I is divided into four parts: the input sizes (the number of nodes, segments, and facets) are reported in rows $2-4$, they are increasing from left to right; the number of break points and protect points are listed in rows $5-6$, respectively; then the running time statistics in individual steps of the CDT algorithm are given in rows $7-11$, the time is reported in seconds; and the total run time (which is the sum of the detailed times) is given in row 12.

From Table I we see that the majority Steiner points of these examples are inserted in the segment recovery step (line 10). Hence this step took the most of the run time. The run times for facet recovery (line 11) were relatively small, because the number of missing facets was small.

Our CDT algorithm will insert Steiner points on the boundary (segments and facets) of the input PLS. In Section 7 we discussed a post-processing step to remove Steiner points from boundaries so that the output mesh can conform to the input surface mesh of the PLS with possibly some Steiner points remaining inside the PLS. Table II reports the experiments of this step on some selected examples. The majority of the Steiner points can be completely deleted from the mesh, only few of them have been relocated to the interior of the domain.

Although our algorithm does not directly work on smooth domains (boundaries are curved surfaces), it is possible to combine our algorithm with other surface meshing algorithms, e.g., [52, 53], such that smooth domains and piecewise smooth domains can be approximated by CDTs. It has been shown that any C^2-smooth surface Σ can be well-approximated by a

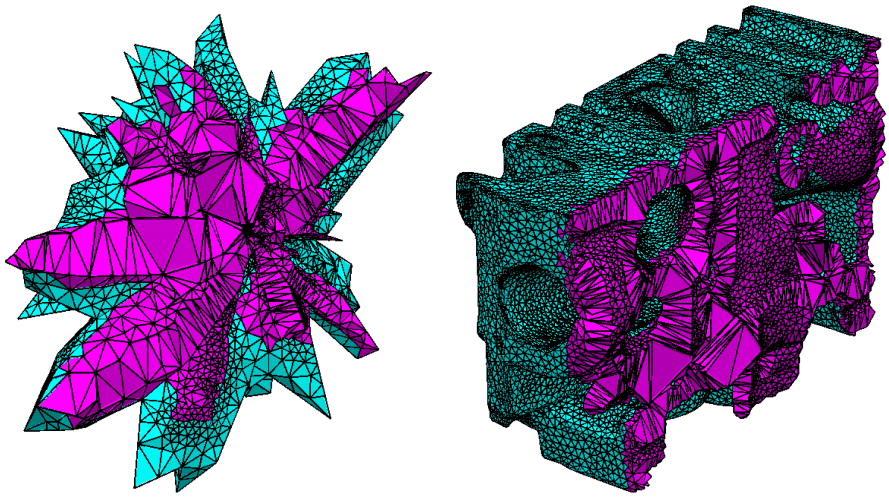


Figure 21. Examples: `Crystal` (left) and `IFP` (right). The generated CDTs are shown. See Table I for their statistics.

	`Cami1a`	`Heart`	`Crystal`	`IFP`
Input Steiner points	482	3,622	10,797	1,963
Deleted Steiner points	446	3,605	10,778	1,957
Relocated Steiner points	36	17	21	6
Total time (sec.)	0.76	0.79	0.88	0.27

Table II. Post-processing: Steiner point removal and relocation (discussed in Section 7).

restricted Delaunay triangulation of a set of ϵ-samples on Σ [54]. Fig. 22 left shows a restricted Delaunay triangulation of a piecewise smooth surface. This model is freely available from [55]. Since such a triangulation is a two-dimensional PLS, the domain can be approximated by a CDT, see Fig. 22. Theoretically, our CDT algorithm needs no Steiner points in constructing the CDTs from such inputs.

9. CONCLUSION

In this paper, the problem of 3D boundary conformity is studied. A challenging question is: Given any 3D polyhedron, can we tetrahedralize it with the number of additional points (Steiner points) as small as possible? This question still remains theoretically open, but some algorithmic improvements have been reached by this work.

We proposed an algorithm to construct a constrained Delaunay tetrahedralization (CDT) of an arbitrary 3D piecewise linear system (PLS). The correctness of this algorithm was theoretically proven. The choices of Steiner points are adaptively determined by local geometric information. The number of Steiner points could be reduced compared to Shewchuk's approach [20]. An analysis on this algorithm was provided. This algorithm is simple and easy to implement. The work and status are summarized as follows:

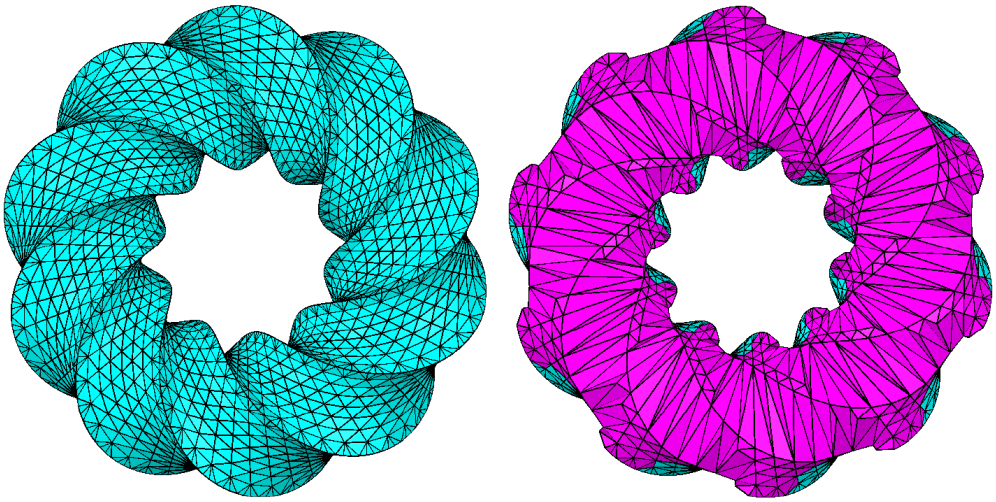


Figure 22. Examples `n10`. Left: a restricted Delaunay triangulation of a piecewise smooth surface. Right: a CDT partition of the domain bounded by this surface (a cut views the interior tetrahedra), it contains no Steiner points.

- We gave a definition of a *constrained Delaunay triangulation* (CDT) of a piecewise linear system (PLS) of any dimension. Our definition allows *Steiner points* (points which do not belong to the input PLS) in a CDT. Hence every PLS can have a CDT. We showed several basic properties of such objects which are close to those of Delaunay triangulations.
- We proposed a practical algorithm for constructing a CDT of any three-dimensional PLS. Steiner points are used in order to recover the boundaries. Termination and correctness of this algorithm are proved. We give a partial analysis of the complexity of the individual steps of the algorithm. It has been implemented and the practical performance is reported through various examples.
- The three segment splitting rules handle the small input angle problem well. Previous works [20, 56, 57] require a separate step to protect the sharp corners, and it must be done in advance. Our rules avoid the pre-processing step by adaptively selecting the segments to be split. Moreover, the achieved edge lengths are usually better, i.e., some input segments may not be split.

Some theoretical questions regarding this algorithm that should be investigated:

- The complexity of the segment recovery algorithm with regard to the input size is not known yet. This is closely related to an open question [29] in computational geometry, i.e., what is the upper bound of the number of Steiner points needed to construct a conforming Delaunay tetrahedralization of a three-dimensional PLS?
- Limiting the number of Steiner points is important since it directly influences the performance of the facet recovery algorithm. Some technical details of the segment recovery algorithm need further investigation. Currently, the segments are split in a randomized order. It generally works well, but sometimes it results in unnecessary Steiner points.

- Although the problem of removing Steiner points from the boundary is discussed and implemented, the problem is far from being solved. Difficulties are arising in practice when the input boundary of the PLS contains anisotropic features.
- The proposed CDT algorithm only takes a piecewise linear boundary as input. It would be a possible extension to let the algorithm directly handle inputs containing smooth surfaces. The *piecewise smooth complex* [53] could be considered.

ACKNOWLEDGEMENTS

This work was supported in part by the WIAS research project `Numerical Mthods` and by the `3D-Topografie` (RGI-011) project led by Delft University of Technology. We thank the referees for their valuable comments and suggestions.

REFERENCES

1. Lennes NJ. Theorems on the simple finite polygon and polyhedron. *American Journal of Mathematics* 1911; **33**(1/4):37–62.
2. Schönhardt E. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen* 1928; **98**:309–312.
3. Ruppert J, Seidel R. On the difficulty of triangulating three-dimensional non-convex polyhedra. *Discrete and Computational Geometry* 1992; **7**:227–253.
4. Chazelle B. Convex partition of a polyhedra: a lower bound and worest-case optimal algorithm. *SIAM Journal on Computing* 1984; **13**(3):488–507.
5. George PL, Hecht F, Saltel E. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering* 1991; **92**:269–288.
6. Weatherill NP, Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering* 1994; **37**:2005–2039.
7. George PL, Borouchaki H, Saltel E. Ultimate robustness in meshing an arbitrary polyhedron. *International Journal for Numerical Methods in Engineering* 2003; **58**:1061–1089.
8. Chazelle B, Palios L. Triangulating a nonconvex polytope. *Discrete and Computational Geometry* 1990; **5**:505–526.
9. Dey TK. Triangulating and CSG representation of polyhedra with arbitrary genus. *Proc. 7th annual ACM Symposium on Computational Geometry*, 1991; 364–372.
10. Chazelle B, Shouraboura N. Bounds on the size of tetrahedralizations. *Discrete and Computational Geometry* 1995; **14**:429–444.
11. Joe B. Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *International Journal for Numerical Methods in Engineering* 1994; **37**:693–713.
12. Hershberger JE, Snoeyink J. Erased arrangements of lines and convex decompositions of polyhedra. *Computational Geometry* 1998; **9**(3):129–143.
13. Edelsbrunner H. *Algorithms in combinatorial geometry*. Springer-Verlag: Heidelberg, 1987.
14. De Berg M, Van Kreveld M, Overmars M, Schwarzkopf O. *Computational Geometry: Algorithms and Applications*. Springer-Verlag: Heidelberg, 1997.
15. Pébay P, Frey PJ. A priori Delaunay-conformity. *Proc. 7th International Meshing Roundtable*, Sandia National Laboratories, 1998; 321–333.
16. Murphy M, Mount DM, Gable CW. A point-placement strategy for conforming Delaunay tetrahedralizations. *Proc. 11th annual ACM-SIAM Symposium on Discrete Algorithms*, 2000; 67–74.
17. Cohen-Steiner D, De Verdière EC, Yvinec M. Conforming Delaunay triangulation in 3D. *Proc. 18th annual ACM Symposium on Computational Geometry*, 2002.
18. Lee DT, Lin AK. Generalized Delaunay triangulations for planar graphs. *Discrete and Computational Geometry* 1986; **1**:201–217.
19. Chew PL. Guaranteed-quality triangular meshes. *Technical Report TR 89-983*, Department of Computer Science, Cornell University 1989.

20. Shewchuk JR. Constrained Delaunay tetrahedralization and provably good boundary recovery. *Proc. 11th International Meshing Roundtable*, Sandia National Laboratories, 2002; 193–204.
21. Si H, Gärtner K. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. *Proc. 14th International Meshing Roundtable*, 2005; 147–163.
22. Baker T. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers* 1989; **5**:161–175.
23. Hazlewood C. Approximating constrained tetrahedralizations. *Computer Aided Geometric Design* 1993; **10**:67–87.
24. Si H. Adaptive tetrahedral mesh generation by constrained Delaunay refinement. *International Journal for Numerical Methods in Engineering* 2008; **75**(7):856–880.
25. Shewchuk JR. Tetrahedral mesh generation by Delaunay refinement. *Proc. 14th Annual Symposium on Computational Geometry*, Minneapolis, Minnesota, United States, 1998; 86–95.
26. Si H. Three dimensional boundary conforming Delaunay mesh generation. PhD Thesis, Institute of Mathematics, Technische Universität Berlin 2008. `http://opus.kobv.de/tuberlin/volltexte/2008/1966/`.
27. Shewchuk JR. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. *Proc. 19th Annual Symposium on Computational Geometry*, 2003; 181–190.
28. Shewchuk JR. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. *Proc. 14th Annual Symposium on Computational Geometry*, 1998; 76–85.
29. Edelsbrunner H. *Geometry and topology for mesh generation*. Cambridge University Press: England, 2001.
30. Miller GL, Talmor D, Teng SH, Walkington NJ, Wang H. Control volume meshes using sphere packing: Generation, refinement and coarsening. *Proc. 5th International Meshing Roundtable*, Sandia National Laboratories, 1996.
31. Huang W. Measuring mesh qualities and application to variational mesh adaptation. *SIAM Journal on Scientific Computing* 2005; **26**(5):1643–1666.
32. Delaunay BN. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 1934; **7**:793–800.
33. Rajan VT. Optimality of the Delaunay triangulation in $\mathbb{R}^d$. *Discrete and Computational Geometry* 1994; **12**:189–202.
34. Clarkson KL, Shor PW. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry* 1989; **4**:387–421.
35. Bowyer A. Computing Dirichlet tessellations. *Comp. Journal* 1987; **24**(2):162–166.
36. Watson DF. Computing the n-dimensional Delaunay tessellations with application to Voronoi polytopes. *Comput. Journal* 1987; **24**(2):167–172.
37. Edelsbrunner H, Shah NR. Incremental topological flipping works for regular triangulations. *Algorithmica* 1996; **15**:223–241.
38. Chan TM, Snoeyink J. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete and Computational Geometry* 1997; **18**(4):433–454.
39. Edelsbrunner H, Tan TS. An upper bound for conforming Delaunay triangulations. *SIAM Journal on Computing* 1993; **22**:527–551.
40. Chew PL. Constrained Delaunay triangulation. *Algorithmica* 1989; **4**:97–108.
41. Shewchuk JR. General-dimensional constrained Delaunay and constrained regular triangulations I: Combinatorial properties. *Discrete and Computational Geometry* 2008; **39**:580–637.
42. Edelsbrunner H. An acyclicity theorem for cell complex in d dimension. *Combinatorica* 1990; **10**(3):251–260.
43. Edelsbrunner H, Mücke M. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithm. *ACM Transactions on Graphics* 1990; **9**(1):66–104.
44. Devillers O, Teillaud M. Perturbations and vertex removal in Delaunay and regular 3D triangulations. *Technical Report 5968*, INRIA 2006.
45. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms* 1995; **18**(3):548–585.
46. Ollivier-Gooch CF. Coarsening unstructured meshes by edge contraction. *International Journal for Numerical Methods in Engineering* 2003; **57**:391–414.
47. Liu A, Baida M. How far flipping can go towards 3D conforming/constrained triangulation. *Proc. 9th International Meshing Roundtable*, Sandia National Laboratories, 2000; 307–315.
48. Freitag L, Ollivier-Gooch CF. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering* 1997; **40**(21):3979–4002.
49. Klingner BM, Shewchuk JR. Aggressive tetrahedral mesh improvement. *Proc. 16th International Meshing Roundtable*, Sandia National Laboratories, 2007; 3–23.
50. Si H. TetGen. `http://tetgen.berlios.de` 2007.
51. INRIA. GAMMA, automatic mesh generation and adaption methods. `http://www-c.inria.fr/gamma`

2007.
52. Boissonnat JD, Oudout S. Provably good surface sampling and meshing of surfaces. *Graphical Models* 2005; **67**:405–451.
53. Cheng SW, Dey TK, Ramos EA. Delaunay refinement for piecewise smooth complexes. *Proc. 18th annual ACM-SIAM Symposium on Discrete Algorithms*, 2007; 1096–1105.
54. Amenta N, Bern M. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry* 1999; **22**:481–504.
55. Edelsbrunner H. 180 wrapped tubes. `http://www.cs.duke.edu/ edels/Tubes`.
56. Cheng SW, Dey TK, Ramos EA, Ray T. Quality meshing for polyhedra with small angles. *International Journal on Computational Geometry and Applications* 2005; **15**:421–461.
57. Pav SE, Walkington NJ. Robust three dimensional Delaunay refinement. *Proc. 13th International Meshing Roundtable*, Sandia National Laboratories, 2004.