3D Boundary Recovery by Constrained Delaunay Tetrahedralization

Journal:	*International Journal for Numerical Methods in Engineering*
Manuscript ID:	NME-Mar-08-0165.R1
Wiley - Manuscript type:	Research Article
Date Submitted by the Author:	21-Aug-2008
Complete List of Authors:	Si, Hang; WIAS, FG3 Gaernter, Klaus; WIAS, FG3
Keywords:	tetrahedral mesh generation, 3D boundary recovery, constrained Delaunay, Steiner points

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2000; **00**:1–6 *Prepared using* nmeauth.cls *[Version: 2002/09/18 v2.02]*

3D Boundary Recovery by Constrained Delaunay Tetrahedralization

H. Si* and K. Gärtner

Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, 10117, Berlin, Germany

SUMMARY

Three-dimensional boundary recovery is a fundamental problem in mesh generation. Theoretical questions of this problem like complexity, optimality, and output size are either NP-complete or still open. 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) in three-dimensional space. Additional points (so-called *Steiner points*) on the constraints are allowed in order to form the CDT. All Steiner points can be removed from the constraints by a post-processing step, few of them may remain in the interior of the mesh. The complexity of this algorithm is discussed. The proposed algorithm has been implemented. The performance of the algorithm is reported through various application examples. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: tetrahedral mesh generation, 3D boundary recovery, constrained Delaunay tetrahedralization, Steiner points

1. INTRODUCTION

Given a domain Ω in $\mathbb{R}^3$, a fundamental problem is how to partition Ω into a set of simple cells such that the boundary $\partial\Omega$ is represented by a union of the cells. This problem is also known as *boundary conformity*. Many applications are based on it.

This problem has many difficulties in three-dimensional space. It is known that additional points (so-called *Steiner points*) are necessary in order to form a tetrahedralization of a polyhedron [1, 2]. See Fig. 1 for examples. Moreover, the problem to determine whether a simple polyhedron can be tetrahedralized without Steiner points is NP-complete [3]. When Steiner points are used, Chazelle [4] showed there are some polyhedra which may require a large number of Steiner points to be tetrahedralized, see Fig. 1 right.

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 mesh exactly. This requires that Steiner points should not be added on the input boundary but only occasionally be added into the interior space. A number of constructive methods have

*Correspondence to: Hang Si, Weierstrass Institute for Applied Analysis and Stochastics (WIAS), Mohrenstrasse 39, 10117, Berlin, Germany

Received March 8, 2008
Revised August 21, 2008

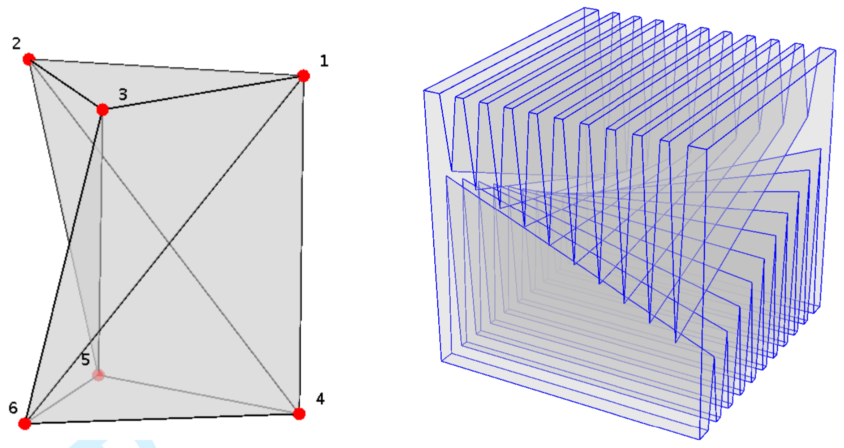


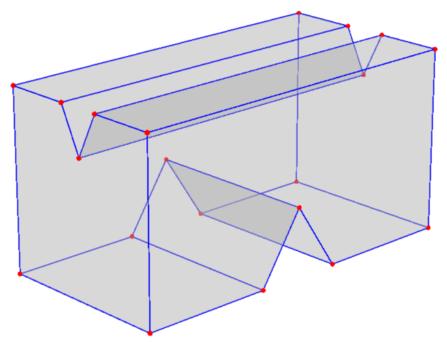
Figure 1. Two polyhedra which are not tetrahedralizable without Steiner point. Left: The Schönhardt polyhedron [2] can be obtained by twisting the upper face around the axes of a parallel triangular prism by a small angle. Right: The Chazelle's polyhedron [4] formed by cutting wedges from a cube. In 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.

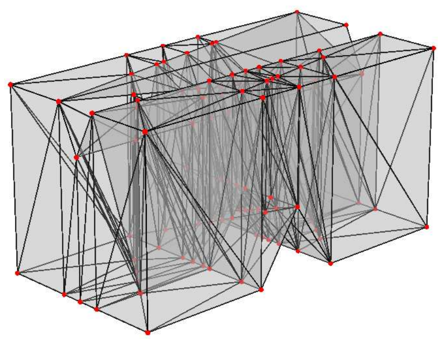
been developed for this purpose [5, 6, 7, 8]. These methods are successful in solving engineering problems. However, many theoretical questions, like the number of Steiner points, the mesh quality, and the complexity, are not well addressed.

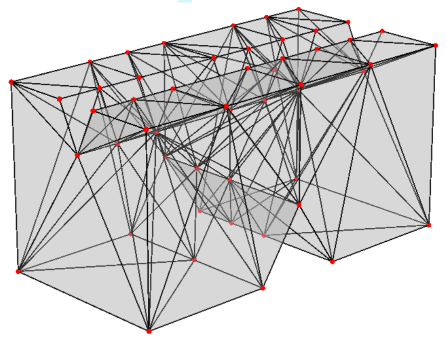
Convex decomposition studies the problem of how to decompose a polyhedron into a set of non-overlapping convex pieces [4, 9, 10, 11, 12, 13, 14]. By allowing Steiner points, Chazelle showed that any simple polyhedron (which has genus 0) P of n vertices can be partitioned into $O(n^2)$ tetrahedra [4]. The bound is tight in the worst case. Call an edge of P *reflex* if its adjacent faces forming an angle larger than 180°. Chazelle and Palios [9] presented an algorithm to decompose a simple polyhedron with n vertices and r reflex edges using $O(n + r^2)$ Steiner points. The main contribution of these works has been to establish the quadratic complexity of the worst case of this problem. However, the proposed algorithms are usually very complicated and will result unnecessarily large number of output tetrahedra. See Fig. 2 (b) for an example. Hence these works are only of theoretical interest.

The Delaunay triangulations are well-studied objects in computational geometry, see, e.g., [15, 16]. A *conforming Delaunay triangulation* of a polyhedron P is a partition of the polyhedron into a set of Delaunay simplices [17], see Fig. 2 (c) for an example. An alternative approach for the boundary conformity problem is to directly enrich the vertex set V of a polyhedron P by adding Steiner points on the boundary ∂P until all simplices on ∂P satisfy the Delaunay criterion [17]. Hence the Delaunay triangulation of V is a conforming Delaunay triangulation of P [18, 19, 20]. This approach, however, may need an unnecessarily large number of Steiner points. Edelsbrunner and Tan [21] provides a cubic upper bounds on the number of Steiner points necessary for a conforming Delaunay triangulation of a two-dimensional polygon. No polynomial upper bound on the number of Steiner points is known in three dimensions.

Constrained Delaunay triangulations are first studied by Lee and Lin [22] and Chew [23] for


(a) The input polyhedron
20 nodes, 12 facets


(b) Convex decomposition
138 nodes, 280 tets


(c) Conforming Delaunay
51 nodes, 103 tets

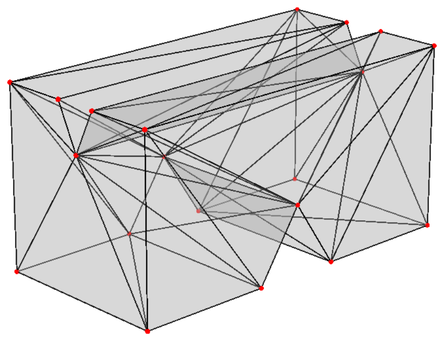

(d) Constrained Delaunay
20 nodes, 29 tets

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

generating two-dimensional Delaunay-like triangulations from planar stright line graphs, see Fig. 3 for an example. The same concept can be generalized into three and higher dimensions. Constrained Delaunay triangulations usually need fewer number of Steiner points than that needed by conforming Delaunay triangulations (see Fig. 2 (d) for an example), and they have many optimal properties as close as those of Delaunay triangulations [24]. Furthermore, many engineering meshing methods [25, 26, 27, 28] make use of them as the intermediate meshes to obtain good quality tetrahedral meshes suitable for numerical simulation.

Algorithms for generating constrained Delaunay tetrahedralizations are proposed [29, 30, 31]. All the algorithms use Steiner points to enforce the boundary conformity. A key question is how to choose Steiner points. In [29], the majority of Steiner points are chosen at the midpoint of missing boundary edges. This algorithm may create many unnecessary Steiner points. A set of Steiner points insertion rules are introduced in [31], such that the Steiner points are chosen adaptively according to the geometric neighbor informations.

Once all boundary edges of P are recovered, Shewchuk shows that it is possible to recover the boundary faces of P without using Steiner points [32]. Another key question is how to efficiently recover facets of P? So far, Shewchuk proposed several algorithms for this purpose [29, 30]. Among them, the flip-based facet insertion algorithm [30] has better performance. In [31], a

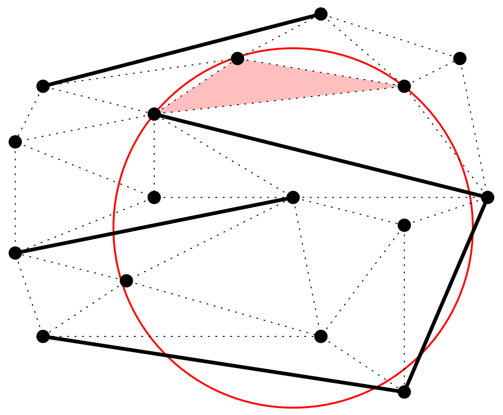


Figure 3. Constrained Delaunay triangulation. The circumcircle of the shaded triangle is not empty but no enclosed vertex is visible from its interior.

simple incremental facet recovery algorithm is proposed.

Our contributions In this paper, we present an theoretical algorithm for constructing constrained Delaunay tetrahedralizations and prove its correctness. This algorithm is based on our previous work [31] and it contains many substantial improvements. First of all, the presented algorithm is greatly simplified over the original algorithm and is easy to implement. We will show that the phase of local degeneracy removal is not required. The segment recovery and facet recovery phases in [31] are simplified and improved. We will compare our segment recovery algorithm with other algorithm. In the facet recovery phase, we will prove that the verification of cavity step in [31] (which is very time consuming) is not needed. The complexity of the proposed algorithm will be analyzed. We further discuss a post-processing step for deleting and suppressing Steiner points from the boundary of the domain.

Outline The rest of the paper is organized as follows. We first formalize the meshing problems in Section 2. In Section 3 we give the definition of constrained Delaunay tetrahedralizations and show some basic properties they have. The proposed algorithm is described in Section 4. The individual phases (segment recovery and facet recovery) of the algorithm are discussed in the consecutive sections, i.e., Section 5, 6, and 7. The analysis of the algorithms are given therein. We discuss the approach for deleting and suppressing Steiner points in Section 8. Some experimental results from publicly available examples are shown in Section 9. We end this paper with a conclusion and an outlook in Section 10.

2. THE MESHING PROBLEMS

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

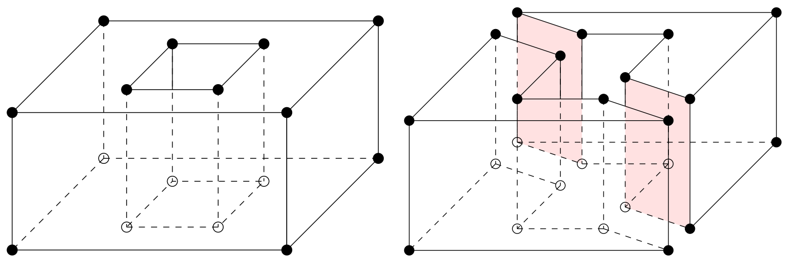


Figure 4. Polyhedra and faces. Left: A polyhedron (a handle) formed by the union of four convex polyhedra. Two faces (one at top and one at bottom) of it are not simply connected. Right: Two polyhedra. The shaded area highlights two 2-faces whose points have the same face figures.

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, $P = \bigcup \mathcal{P}$, 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 $\mathcal{P}$.

We modify the suggestion of Edelsbrunner [33] to define faces of P. 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* $\text{aff}(F)$, i.e., $\dim(F) = \dim(\text{aff}(F))$. 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* $\text{bd}(P)$ of P. The *interior* of P is $\text{int}(P) = P - \text{bd}(P)$.

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) $P, Q \in \mathcal{X} \Longrightarrow P \cap Q \subset \mathcal{X}$, 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.* [34] 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 subset of $\mathcal{X}$. *(iii)* is an extra property for a PLS which makes it more flexible. For example, it allows a cube encloses an edge in its interior with no need to further decompose it. Furthermore, it excludes the case which two polyhedra having the same dimension 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*

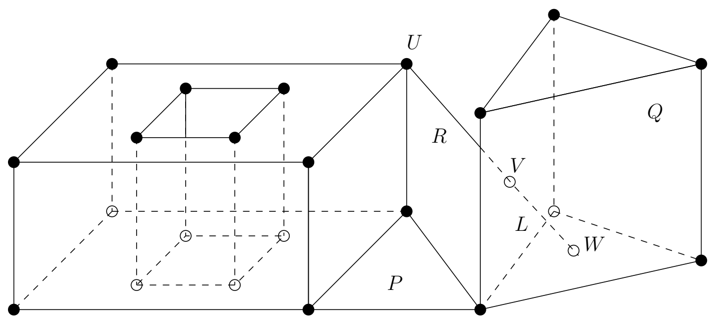


Figure 5. A three-dimensional piecewise linear system $\mathcal{X}$. In this figure, the dimensions of polytopes $Q, P, R, L \in \mathcal{X}$ are $3, 2, 1, 1$, respectively. $U, V, W \in \mathcal{X}$ are 0-polytopes, where $V = R \cap L = R \cap Q$, $L = L \cap Q$, and $\dim(L) < \dim(Q)$.

set, $\text{vert}(\mathcal{X})$, 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 [35].

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 $|\mathcal{K}| = P$.

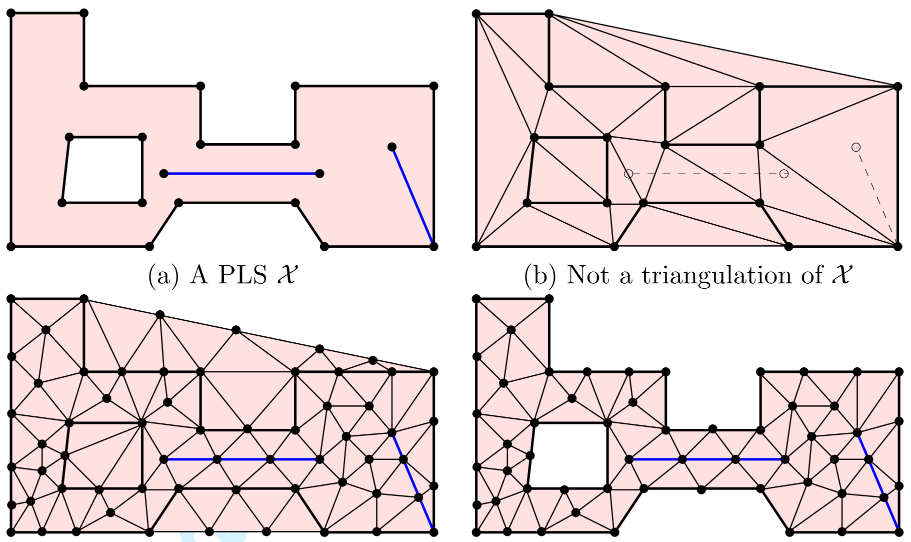
Note that $\mathcal{T}$ may contain Steiner points. We define a *mesh* of $\mathcal{X}$ to be a subcomplex $\mathcal{K}$ of $\mathcal{T}$ such that $|\mathcal{K}| = |\mathcal{X}|$. According to our definitions, a triangulation of a set S of vertices triangulates the convex hull of S, while a mesh of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

The meshing problems Let $\mathcal{X}$ be a three-dimensional PLS. The *three-dimensional boundary conformity problem* is: Find a tetrahedral mesh $\mathcal{T}$ of $\mathcal{X}$ such that

(1) the number of Steiner points in $\mathcal{T}$ is bounded, and
(2) the mesh quality of $\mathcal{T}$ is optimal.

The first requirement means that the number of Steiner points in $\mathcal{T}$ should be limited as small as possible. One can achieve it by showing a polynomial upper bound on the number of Steiner points depending only on the input size of $\mathcal{X}$.

The requirement *(2)* needs more definitions. First of all, the definition of mesh quality depends on applications. In the context of numerical simulations, the mesh quality is be determined together by several measures on element shape, size, and orientation [36]. In general, the mesh quality can be expressed by an *object function* $f : \mathcal{P}_T \rightarrow \mathbb{R}$, where $\mathcal{P}_T$ is the collection of all meshes of $\mathcal{X}$, and $f(\mathcal{T})$ maps a $\mathcal{T} \in \mathcal{P}_T$ into a real value. The *optimal mesh* is defined to be the one on which f attains its minimum. For an example, when the mesh is used



(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.

for function approximation, then the Delaunay triangulation minimizes the interpolation error for the lifting function $\|\mathbf{x}\|^2$ among all other triangulations of the same vertex set [37, 38].

Among different mesh quality measures, 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 the existing vertices. In other words, the shortest edge length in $\mathcal{T}$ can be bounded by some constant divided by the smallest feature size of $\mathcal{X}$.

We further consider a *stronger* three-dimensional boundary conformity problem: Let $\mathcal{F}$ be a (surface) mesh of $\partial\mathcal{X}$, find a tetrahedral mesh $\mathcal{T}$ of $\mathcal{X}$ such that $\mathcal{T}$ satisfies the above two requirements, in additional, $\mathcal{F}$ is a subcomplex of $\mathcal{T}$. In other words, the stronger meshing problem forbiddens the adding of Steiner points on the input boundary.

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 criterion* (also known as the "empty sphere" criterion) [17]: 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 [39] which are useful in various applications. Efficient algorithms are proposed for computing Delaunay triangulations in two and three dimensions, see e.g., [40, 41, 42, 43, 44].

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 is usually needed in a conforming Delaunay triangulation of $\mathcal{X}$. Edelsbrunner *et al* [21] proved that one need at most $O(n^3)$ Steiner points for obtaining a conforming Delaunay triangulation in $\mathbb{R}^2$.

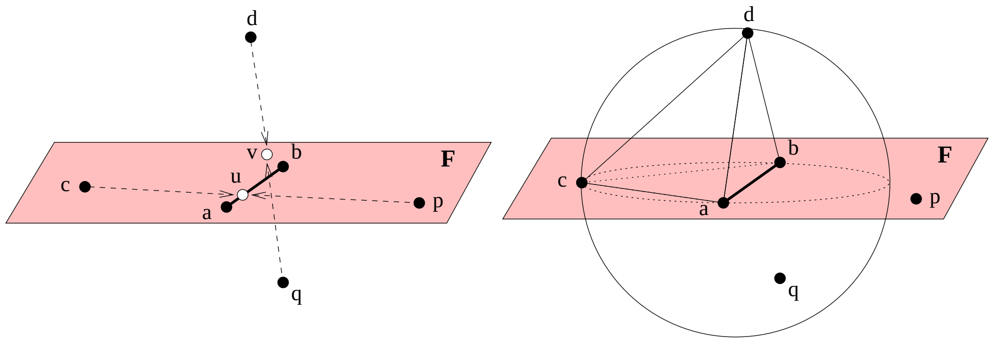


Figure 7. Visibility and constrained Delaunay criterion. The shaded region is a facet F of a PLS $\mathcal{X}$ in $\mathbb{R}^3$, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{p} \in F$. $\mathbf{ab}$ is a segment of $\mathcal{X}$. Left: $\mathbf{d}$ and $\mathbf{q}$ are invisible to each other since $\mathbf{dq} \cap F = \mathbf{v}$. $\mathbf{c}$ 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 circumball 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.

No upper bound is available yet in dimension higher than two.

Let $\mathcal{G}$ be a planar straight line graph. Any two edges of $\mathcal{G}$ is 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 [22] and Chew [23]. 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. Two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are *invisible* to each other if the interior of the line segment $\mathbf{xy}$ intersects a polyhedron $P \in \mathcal{X}$ 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 crietrion*, 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 $\mathbb{R}^d$ 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 circumball B_σ of σ which is empty.
(ii) There exists $F \in \mathcal{X}$, such that $\text{int}(\sigma) \subseteq \text{int}(F)$. Let $K = S \cap \text{aff}(F)$, then no vertex of K inside B_σ is visible from any point in $\text{int}(\sigma)$.

Case *(i)* means that every Delaunay simplex is constrained Delaunay. In *(ii)*, F is the lowest-dimensional polyhedron of $\mathcal{X}$ that contains σ, K is the subset of S in the affine hull generated by F. A simplex $\sigma \subset F$ is constrained Delaunay or not only depends on vertices of K. See Fig. 7 right for examples.

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 [22] and Chew [45]. Shewchuk's definition of a CDT [24] is also a pure CDT. It is well known that a pure CDT of a 3-dimensional PLS may not exist while there are infinitely many CDTs of $\mathcal{X}$ with Steiner points.

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 proofs are omitted, they are found in [35].

Delaunay triangulations can be checked locally. It is true for CDTs as well. Let $\mathcal{X}$ be a d-dimensional PLS. Let $\mathcal{T}$ be any triangulation of $\mathcal{X}$. A $(d-1)$-simplex σ of $\mathcal{T}$ is *locally Delaunay* if either *(i)* σ is on the convex hull, or *(ii)* $\sigma \subset |\partial \mathcal{X}|$, or *(iii)* the opposite vertex of τ is not in $\text{int}(B_\nu)$ of ν, where $\tau, \nu \in \mathcal{T}$ and $\sigma = \tau \cap \nu$. Note that *(ii)* implies that one can ignore the $(d-1)$-simplices contained in the boundary of $|\mathcal{X}|$.

Theorem 1 (Constrained Delaunay Lemma [35]) *If every $(d-1)$-simplex of $\mathcal{T}$ is locally Delaunay, then $\mathcal{T}$ is a CDT of $\mathcal{X}$.* ■

If a point set S in $\mathbb{R}^d$ is in general position, i.e., no $d+2$ points of S share a common $(d-1)$-sphere, then the Delaunay triangulation 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 d-dimensional PLS. The i-skeleton $\mathcal{X}^{(i)}$ of $\mathcal{X}$ is an i-dimensional PLS, where $0 \le i \le d-1$. It is useful to know the properties of a CDT of $\mathcal{X}^{(i)}$.

Theorem 3 ([35]) *Let $\mathcal{X}$ be a d-dimensional PLS.*
(i) *A CDT of $\mathcal{X}^{(d-1)}$ is a CDT of $\mathcal{X}$.*
(ii) *A CDT of $\mathcal{X}^{(i)}$ is a conforming Delaunay triangulation of $\mathcal{X}^{(i)}$, where* $0 \le i \le d-2$. ■

For examples, if $\mathcal{X}$ is a 3-dimensional PLS in $\mathbb{R}^3$, then a CDT of the 1-skeleton of $\mathcal{X}$ (the set of vertices and segments of $\mathcal{X}$) is a conforming Delaunay triangulation. And a CDT of the 2-skeleton of $\mathcal{X}$ (the set of vertices, segments, and facets of $\mathcal{X}$) is also a CDT of $\mathcal{X}$.

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 of $\mathcal{X}$ consists of a set of vertices, segments, and facets. 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 $\mathcal{X}^{(2)}$.

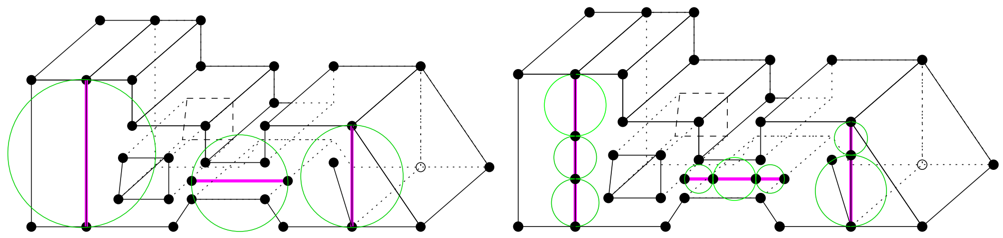


Figure 8. A condition (Theorem 5) guarantees the existence of a CDT without Steiner points. The left PLS does not satisfy the condition. The right PLS whose all edges are Delaunay satisfies the condition if the vertex set of the PLS is in general position.

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 incrementally constructing a CDT $\mathcal{D}_i$ of $\mathcal{X}^{(i)}$ from a CDT $\mathcal{D}_{i-1}$, where $i = 1, 2$. By Theorem 3, $\mathcal{D}_2$ is a CDT of $\mathcal{X}$. A constrained Delaunay mesh of $\mathcal{X}$ can be obtained by removing simplices of $\mathcal{D}_2$ not in $|\mathcal{X}|$.

In this algorithm, Steiner points are only introduced in the step *2* (segment recovery). 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 [46, 29, 47]. Hence, theoretically, there is no need to actually perturb the vertices. We will discuss this issue in the implementation of this algorithm.

Shewchuk [32] 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 [32] 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 [32] which will be satisfied in our algorithm (see Fig. 8).

Once the existence of a CDT with no Steiner points is known, we still need to show that the step *3*, i.e., facet recovery, can be done without using Steiner points. We will postpone the proof of the correctness of the algorithm until the end of Section 6.2.

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*

We need some definitions. A vertex of $\mathcal{X}$ is *acute* if at least two segments of $\mathcal{X}$ incident at it form an angle less than 90°. We distinguish two types of segments in $\mathcal{X}$: a segment is *type-0* if its both endpoints are not acute, it is *type-1* if only one of its endpoints is acute. If both endpoints of a segment are acute, it is treated as a type-0 segment at the beginning and it is transformed into two type-1 segments immediately after a Steiner point is inserted in it.

Let $\mathbf{e}_i\mathbf{e}_j$ be a segment of $\mathcal{X}$ with endpoints $\mathbf{e}_i$ and $\mathbf{e}_j$. $\mathbf{e}_i\mathbf{e}_j$ is split by adding a Steiner point in the interior of it. The two resulting edges are called *subsegments* of $\mathbf{e}_i\mathbf{e}_j$. Subsegments inherit types from the original segments. For example, if $\mathbf{e}_i\mathbf{e}_j$ is a subsegment of $\mathbf{e}_1\mathbf{e}_2$ which is a type-1 segment, $\mathbf{e}_i\mathbf{e}_j$ is also type-1 although none of its endpoints is acute. For any vertex $\mathbf{v}$ inserted on a type-1 segment (or subsegment), let $R(\mathbf{v})$ denote its original acute vertex. A tacit rule is used throughout this section, if $\mathbf{e}_i\mathbf{e}_j$ is a type-1 segment, it implies that either $\mathbf{e}_i$ or $R(\mathbf{e}_i)$ is acute. In the following, unless it is explicitly mentioned, the term "segment" means either a segment or a subsegment.

The *diametric circumball* of a segment is the smallest circumscribed ball of it. A vertex *encroaches upon* a segment if it lies inside the diametric circumball of that segment. We have the following fact about missing segments of $\mathcal{X}$.

Fact 6. *If a segment of $\mathcal{X}$ is missing in $\mathcal{D}_0$ and vert($\mathcal{D}_0$) is in general position, then it must be encroached by at least one vertex of $\mathcal{D}_0$.*

Let $\mathbf{e}_i\mathbf{e}_j$ be a missing segment, it will be split by a vertex $\mathbf{v}$ 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 $\mathbf{e}_i\mathbf{e}_j$. The choice of $\mathbf{v}$ is governed by three rules given below. Let $\Sigma(\mathbf{c}, r)$ be a sphere with cenetr $\mathbf{c}$ and radius r, and let $\|\cdot\|$ be the Euclidean distance function:

```
1. $\mathbf{e}_i\mathbf{e}_j$ 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. $\mathbf{e}_i\mathbf{e}_j$ is type-1 (Fig. 9 middle), let $\mathbf{e}_k = R(\mathbf{e}_i)$, then $\mathbf{v} = \mathbf{e}_k\mathbf{e}_j \cap \Sigma(\mathbf{c}, r)$, where $\mathbf{c} = \mathbf{e}_k$ and
   $r = \|\mathbf{e}_k - \mathbf{p}\|$. However, if $\|\mathbf{v} - \mathbf{e}_j\| < \|\mathbf{v} - \mathbf{p}\|$, 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
   $\mathbf{v} = \mathbf{e}_k\mathbf{e}_j \cap \Sigma(\mathbf{c}, r)$, where $\mathbf{c} = \mathbf{e}_k$, and
     if $\|\mathbf{p} - \mathbf{v}'\| < \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$ then
        $r = \|\mathbf{e}_k - \mathbf{e}_i\| + \|\mathbf{e}_i - \mathbf{v}'\| - \|\mathbf{p} - \mathbf{v}'\|$;
     else
        $r = \|\mathbf{e}_k - \mathbf{e}_i\| + \frac{1}{2}\|\mathbf{e}_i - \mathbf{v}'\|$;
     end.
```

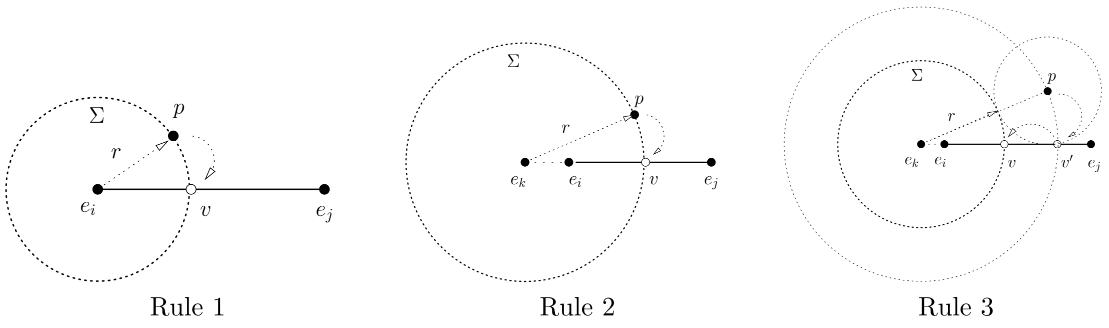


Figure 9. Segment splitting rules.

The idea of these rules is to avoid creating unnecessarily short edges. All the three rules guarantee that the newly inserted vertices do not too close to the existing ones. Note that Rule 1 and 2 never create an edge shorter than the distance $\|R(\mathbf{e}_i) - \mathbf{v}\|$. Rule 3 may create an edge which is at most one third of the length of $\|R(\mathbf{e}_i) - \mathbf{e}_j\|$. Later we will show the total number of applying Rule 3 is bounded.

For several segments sharing an acute vertex, by repeatedly using Rule 2 or 3, a protecting ball is automatically created which ensures: no other vertex can be inserted inside the ball. The effect is shown in Fig. 10 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 below. The inputs are a three-dimensional PLS $\mathcal{X}$ and a Delaunay tetrahedralization $\mathcal{D}_0$ of vert($\mathcal{X}$). The algorithm initializes 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 (line 6). $\mathbf{v}$ 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 $\mathcal{S}$ (line 7). 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 (line 8). Here B_σ means the diametric circumball of a segment $\sigma \in \mathcal{X}^{(1)}$. $\mathcal{D}_1$ is updated to a Delaunay tetrahedralization of the vertex set including $\mathbf{v}$ (line 9).

5.3. *Proof of Termination*

The termination of this algorithm can be proved by showing that the length of every subsegment will not be arbitrarily small. The *local feature size* [48] 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.

Theorem 7. *Let $\mathbf{e}_i\mathbf{e}_j$ be a resulting subsegment.*

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

- *if* $\mathbf{e}_i\mathbf{e}_j$ *is type-1, then:*
  $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \min\{lfs(\mathbf{e}_i), lfs(\mathbf{e}_j)\}$.
- *if* $\mathbf{e}_i\mathbf{e}_j$ *is type-2, let* $\mathbf{e}_k = R(\mathbf{e}_i)$*, then:*
  $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \frac{1}{C} lfs(\mathbf{e}_k)$ *when* $\mathbf{e}_i = \mathbf{e}_k$,
  $\|\mathbf{e}_i - \mathbf{e}_j\| \geq lfs(\mathbf{e}_k)\sin(\theta)$ *when* $\mathbf{e}_i \neq \mathbf{e}_k$.

where C *is a bounded constant and* θ *is the smallest input angle.*

Proof Every inserted vertex has a reference point which is responsible for the insertion. Let $\mathbf{v}$ be an inserted vertex, and $P(\mathbf{v})$ be its reference point.

If $\mathbf{e}_i\mathbf{e}_j$ is type-1. Without loss of generality, let $\mathbf{e}_i$ be an inserted vertex, then $\mathbf{p}_i = P(\mathbf{e}_i)$ is either an input vertex or an inserted vertex on a segment which is not incident with $\mathbf{e}_i\mathbf{e}_j$. By Rule 1 $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \|\mathbf{e}_i - \mathbf{p}_i\| \geq \text{lfs}(\mathbf{e}_i)$.

If $\mathbf{e}_i\mathbf{e}_j$ is type-2 and let $\mathbf{e}_k = R(\mathbf{e}_i)$. If $\mathbf{e}_i = \mathbf{e}_k$ ($\mathbf{e}_i$ is the acute vertex), let $\mathbf{e}_i\mathbf{e}_s$ be the segment containing $\mathbf{e}_i\mathbf{e}_j$. Without loss of generality, assume there are n segments incident at $\mathbf{e}_i$ (including $\mathbf{e}_i\mathbf{e}_s$), and let $\mathbf{e}_i\mathbf{e}_a$ be the shortest segment among them, then $\text{lfs}(\mathbf{e}_i) \leq \|\mathbf{e}_i - \mathbf{e}_a\|$.

Consider the worst case, that $\mathbf{e}_i\mathbf{e}_a$ is first halved by a Rule 3, and due to multi-encroachment, $\mathbf{e}_i\mathbf{e}_s$ is also be split into $\mathbf{e}_i\mathbf{v}_1$ and $\mathbf{v}_1\mathbf{e}_s$. If $\mathbf{v}_1 = \mathbf{e}_j$, then $\|\mathbf{e}_i - \mathbf{e}_j\| = \frac{1}{2}\|\mathbf{e}_i - \mathbf{e}_a\| \geq \frac{1}{2}\text{lfs}(\mathbf{e}_i)$, else $\mathbf{e}_i\mathbf{v}_1$ will be split again. Assume again the worst case, i.e., $\mathbf{e}_i\mathbf{v}_1$ is halved by a Rule 3 into $\mathbf{e}_i\mathbf{v}_2$ and $\mathbf{v}_2\mathbf{v}_1$. But the cause of inserting $\mathbf{v}_2$ must not due to $\mathbf{e}_i\mathbf{e}_a$ because the incident parts (at $\mathbf{e}_i$) of both segments have been split into the same length. Again consider the worst case, the insertion of $\mathbf{v}_2$ is caused by a split of another segment $\mathbf{e}_i\mathbf{e}_b$. If $\mathbf{v}_2 = \mathbf{e}_j$, then $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \frac{1}{4}\|\mathbf{e}_i - \mathbf{e}_1\| \geq \frac{1}{4}\text{lfs}(\mathbf{e}_i)$. If $\mathbf{e}_i\mathbf{e}_j$ still doesn't appear, the splitting procedure will continue. In the worst case, $\mathbf{e}_i\mathbf{e}_s$ is split at most n times, and each time is originally caused by a Rule 3. Hence $\|\mathbf{e}_i - \mathbf{v}_n\| \geq \frac{1}{2n}\text{lfs}(\mathbf{e}_i)$. If $\mathbf{v}_n = \mathbf{e}_j$, then $C = 2n$.

Now it is possible that $\mathbf{v}_n \neq \mathbf{e}_j$, i.e., $\mathbf{e}_i\mathbf{e}_s$ will be split again. However, after at most n times Rule 3 splits, the new split of $\mathbf{e}_i\mathbf{e}_s$ must not be caused by any incident segment at $\mathbf{e}_i$. Hence the cause of the split is either from a disjoint segment or an existing vertex. Since there are only finite number of input vertices and input segments, C is finite.

If $\mathbf{e}_i \neq \mathbf{e}_k$, then $\mathbf{p} = P(\mathbf{e}_i)$, and $\|\mathbf{e}_i - \mathbf{e}_j\| \geq \|\mathbf{e}_k - \mathbf{e}_i\|\sin(\theta) \geq \text{lfs}(\mathbf{e}_i)\sin(\theta)$ (θ be the angle between $\mathbf{e}_k\mathbf{e}_i$ and $\mathbf{e}_k\mathbf{p}$). ■

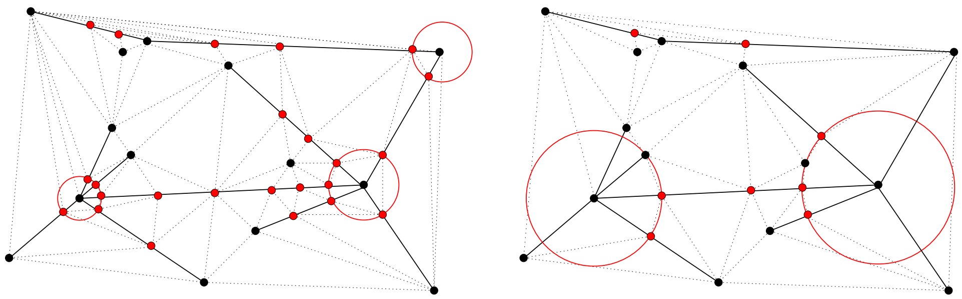


Figure 10. Comparison with other algorithm (in 2D). The Steiner vertices are shown in red. The protecting balls around acute vertices are shown (in red). Left: a result of Shewchuk's algorithm [29], 24 Steiner points. Right: a result of our algorithm, 8 Steiner points.

Theoretically, the constant C in the above theorem could be exponentially large. In our experiments on various data sets, the algorithm terminates within a few steps creating the protection ball sector. C is usually no larger than 4.

5.4. *Comparison with Other Algorithm*

Shewchuk proposed a segment recovery algorithm [29] for the same purpose. This algorothm 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. 10). The radii of these spheres are chosen priori such that they are not unnecessarily small. The subsegments inside the spheres are strongly Delaunay, and no later Steiner points can be inside the spheres. The second step recovers other non-Delaunay (sub)segments by recursively bisection. Fig 10 shows a comparison between the two algorithms in two dimensions. It shows that Shewchuk's algorithm may create many unnecessary Steiner points than Ours.

Next example shows that Shewchuk's algorithm may introduce many unnecessarily short edges. The input PLS is shown in Fig. 11 (a). It consists of $m+1$ segments all meeting at one single point $\mathbf{a}$. Assume that all segments have the same length. An additional point $\mathbf{v}$ lies slightly below the segment $\mathbf{ab}_0$, and let $\|\mathbf{a}-\mathbf{v}\| = \frac{2}{3}\|\mathbf{a}-\mathbf{b}_0\| - \epsilon$, where $0 < \epsilon << \|\mathbf{a}-\mathbf{b}_0\|$. The first step of the algorithm protects $\mathbf{a}$ by adding Steiner points $\mathbf{c}_0, ..., \mathbf{c}_m$ on segments (Shown in (b)) such that $\|\mathbf{a}-\mathbf{c}_m\| = \frac{1}{3}\|\mathbf{a}-\mathbf{b}_m\|$.

Now because of the presence of $\mathbf{v}$, segment $\mathbf{c}_0\mathbf{b}_0$ is non-Delaunay. The second step of this algorithm will bisect $\mathbf{c}_0\mathbf{b}_0$ by adding $\mathbf{u}_0$, and so do for $\mathbf{c}_i\mathbf{b}_i$ by adding $\mathbf{u}_i$, for $i \leq m$. One can easily see $\|\mathbf{c}_m - \mathbf{u}_m\| = 2^{-1}\frac{2}{3}\|\mathbf{a}-\mathbf{b}_m\|$.

The bisection process will continue until all subsegments are Delaunay. The final status is shown in (c). One can deduce that the length:

$$\|\mathbf{w}_m - \mathbf{u}_m\| = 2^{-k}\frac{2}{3}\|\mathbf{a}-\mathbf{b}_m\|,$$

where k is the number of bisections on each segment. In particular,

$$\frac{2}{3}\|\mathbf{a}-\mathbf{b}_m\| \geq \mathrm{lfs}(\mathbf{w}_m).$$

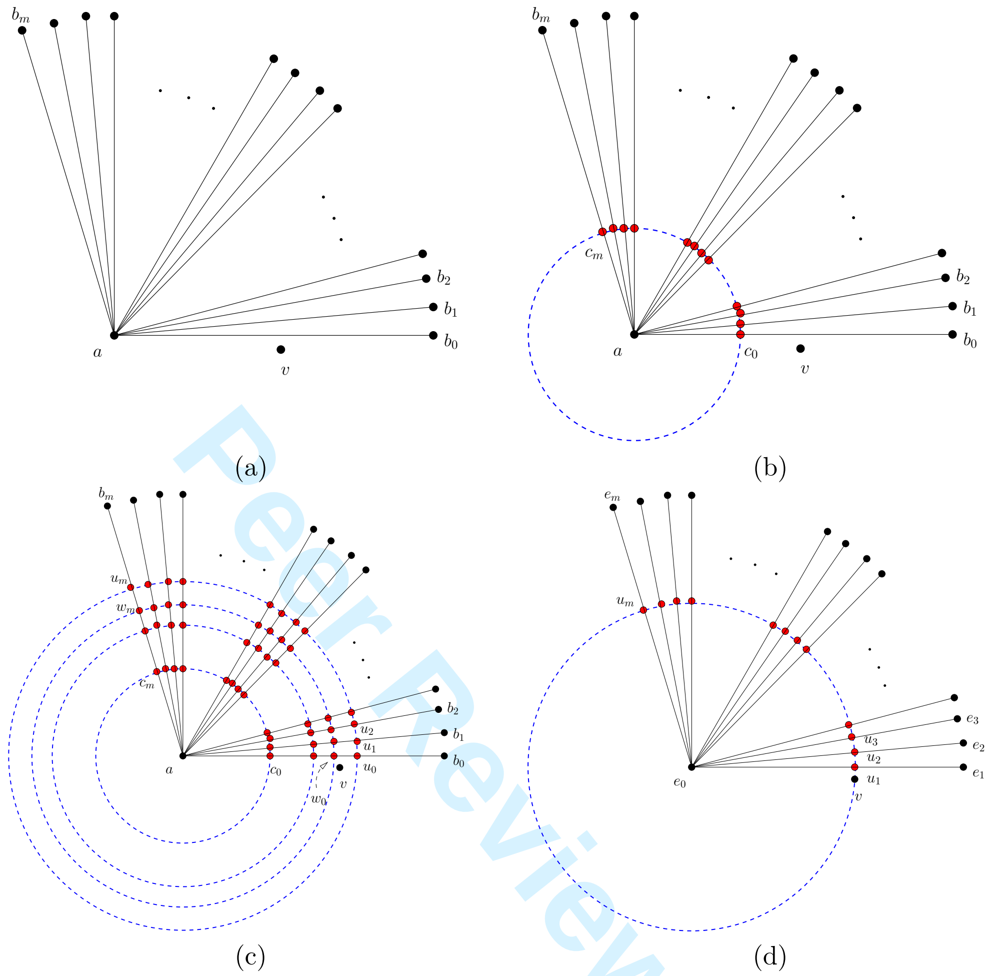


Figure 11. A study of the number of Steiner points may be introduced by the algorithm of Shewchuk [29] and our algorithm.

Put these together, we have

$$\|\mathbf{w}_m - \mathbf{u}_m\| \geq 2^{-k}\,\mathrm{lfs}(\mathbf{w}_m).$$

One can make k arbitrarily large by moving $\mathbf{v}$ arbitrarily close to $\mathbf{ab}_0$. Hence the edge length $\|\mathbf{u}_m - \mathbf{w}_m\|$ may be arbitrarily short. As a result, many short edges (on the circles) and many Steiner points are created. Our algorithm will only create $m+1$ Steiner points (see (d)) on the circle centered at $\mathbf{a}$ with radius $\|\mathbf{a} - \mathbf{v}\|$.

5.5. *Computational Issues*

One frequently used operation in our segment recovery algorithm is to match a segment with an edge of $\mathcal{D}_1$. Let $\mathbf{ab}$ be a segment to be matched. The *star* of $\mathbf{a}$, St($\mathbf{a}$), contains all simplices of $\mathcal{D}_1$ that contain $\mathbf{a}$, and the *link* of $\mathbf{a}$, $\mathrm{Lk}(\mathbf{a})$, contains all faces of simplices in $\mathrm{St}(\mathbf{a})$ that do not contain $\mathbf{a}$. One first locate a tetrahedron $\tau \in \mathrm{St}(\mathbf{a})$, then search $\mathbf{ab}$ in $\mathrm{St}(\mathbf{a})$. If $\mathbf{ab} \in \mathrm{St}(\mathbf{a})$, then $\mathbf{ab}$ is matched in $\mathcal{D}_1$, otherwise, it is missing and will be recovered. The main cost of this operation is the time for locating τ. By available point location algorithms, e.g., [49], the matching of a segment could be done efficiently.

Let $\mathbf{ab}$ be a missing segment. The reference point $\mathbf{p}$ for splitting $\mathbf{ab}$ is randomly chosen from the set of encroaching points of $\mathbf{ab}$. Another frequent operation is to collect the set of encroaching points of $\mathbf{ab}$. This is done by performing a line search from $\mathbf{a}$ towards $\mathbf{b}$. The worst-case time complexity for searching a reference point could be $O(n^2)$, such an example is found in [50]. However, the average time for performing a line searching is much smaller.

Once a Steiner point $\mathbf{v}$ is found (by one of the segment splitting rules), it is inserted into $\mathcal{D}_1$ and $\mathcal{D}_1$ is updated into a Delaunay tetrahedralization including $\mathbf{v}$. The main costs for inserting $\mathbf{v}$ are point location and Delaunay update of $\mathcal{D}_1 \cup \{\mathbf{v}\}$. It can be shown that both steps could be done efficiently.

After the insertions of $\mathbf{v}$, $\mathbf{ab}$ is replaced by two subsegments $\mathbf{av}$ and $\mathbf{vb}$. They are added in $\mathcal{S}$. The whole process is repeated until $\mathcal{S}$ is empty. The complexity of the segment recovery algorithm depends on the total number of Steiner points n_s. In the worst-case, the algorithm may need $O(n_s^2 \log n_s)$ time.

The robustness of this implementation takes advantage of the fact that $\mathcal{D}_1$ is always a Delaunay tetrahedralization. The updating of $\mathcal{D}_1 \cup \{\mathbf{v}\}$ incrementally builds a new Delaunay tetrahedralization. Hence high precision or exact point-in-sphere test can be performed, see e.g. [51, 52].

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 $\mathcal{D}_1$. The Assumption 4, i.e., the vertex set of $\mathcal{D}_1$ is in general position, is important. It guarantees that the facets of $\mathcal{X}$ can be recovered without using Steiner points.

6.1. *The Algorithm*

Each facet $F \in \mathcal{X}$ together with the Steiner points inserted on F is first triangulated into a two-dimensional CDT $\mathcal{T}_F$. Hence $\partial\mathcal{X}$ is triangulated into a triangulation $\mathcal{F}$. We call triangles of $\mathcal{F}$ *subfaces* to distinguish other faces of $\mathcal{D}_2$. Some subfaces may be missing in $\mathcal{D}_2$. The facet recovery algorithm incrementally recover missing subfaces of $\mathcal{F}$.

At initialization, let $\mathcal{D}_2 = \mathcal{D}_1$; add all missing subfaces of $\mathcal{F}$ into a set $\mathcal{S}$. The algorithm iteratively recovers the subfaces in $\mathcal{S}$ and update $\mathcal{D}_2$, it stops when $\mathcal{S}$ is empty.

At each iteration i, several missing subfaces are recovered together. We define a *missing region* Ω to be a set of subfaces of $\mathcal{F}$ such that
(i) all subfaces in Ω are coplanar,
(ii) the edges on $\partial\Omega$ are edges of $\mathcal{D}_2$, and
(iii) the edges in $\mathrm{int}(\Omega)$ are missing in $\mathcal{D}_2$.
Hence Ω is a connected set of missing subfaces. It may not be simply connected, i.e., Ω can contain a hole inside. Each missing subface belongs to a missing region. A facet can have more than one missing regions.

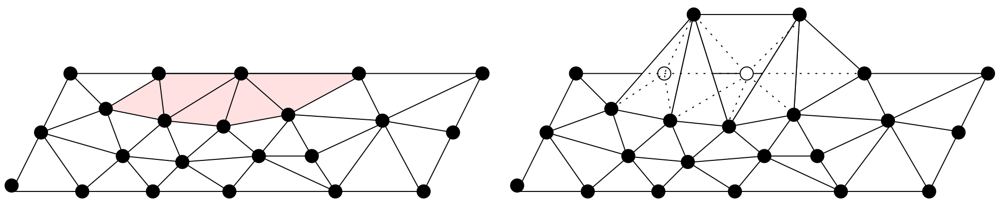


Figure 12. Left: The shaded area highlights a missing region Ω. Right: One of the cavities resulting from a missing region is illustrated.

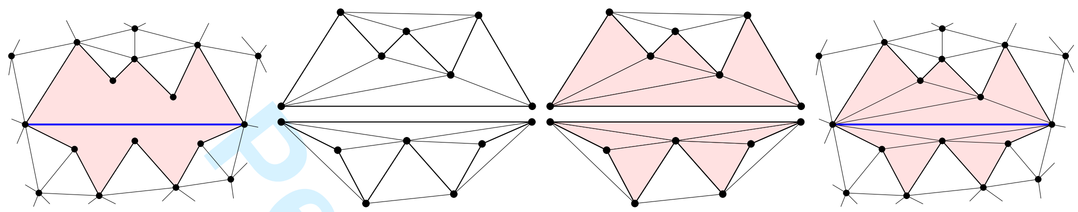


Figure 13. The FacetRecovery algorithm (illustrated in 2D). From left to right, the two initial cavities C_1 and C_2 separated by a segment, the initial Delaunay triangulations $\mathcal{D}_{C_1}$ and $\mathcal{D}_{C_2}$, triangles of $\mathcal{D}_{C_1}$ and $\mathcal{D}_{C_1}$ are classified as "inside" or "outside", and the new partitions of C_1 and C_2.

When a missing region Ω is found, one can derive a cavity in $|\mathcal{D}_2|$ by removing all its 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, some of them are subfaces of $\mathcal{F}$.

The next step is to tetrahedralize each cavity C without using Steiner points. The TetrahedralizeCavity subroutine first constructs the Delaunay tetrahedralization $\mathcal{D}_C$ of vert($\mathcal{C}$) (line 1). Next it removes those tetrahedra of $\mathcal{D}_C$ which are not in the interior of C from $\mathcal{D}_C$ (lines $2-6$). On finish, the remaining tetrahedra in $\mathcal{D}_C$ form a partition of C.

```
Subroutine TetrahedralizeCavity ($C$)
// $C$ is a cavity (a polyhedron with triangular facets).
1.  form the Delaunay tetrahedralization $\mathcal{D}_C$ of vert($C$);
2.  for each tetrahedron $\tau \in \mathcal{D}_C$, do
3.      if $\tau \nsubseteq$ int($C$), then
4.          $\mathcal{D}_C = \mathcal{D}_C \setminus \{\tau\}$;
5.      endif
6.  endfor
7.  return $\mathcal{D}_C$;
```

The FacetRecovery algorithm first initializes a set $\mathcal{S}$ of all subfaces of $\mathcal{X}$. Then it runs into a loop until $\mathcal{S}$ 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 crossing Ω are removed from $\mathcal{D}_2$ (line 7) resulting a temporary object $\mathcal{D}'_2$. Two cavities C_1 and C_2 separated by Ω are formed in the interior of $|\mathcal{D}_2|$ (line 8). 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 9 and 10), respectively. $\mathcal{D}_2$ is

```
Algorithm 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 set $\mathcal{S}$ of all subfaces of $\mathcal{X}$;
 3.  while $\mathcal{S} \neq \emptyset$ do
 4.      get a subface $\sigma \in \mathcal{S}$; $\mathcal{S} = \mathcal{S} \setminus \{\sigma\}$;
 5.      if $\sigma$ is missing in $\mathcal{D}_2$, then
 6.          form a missing region $\Omega$ containing $\sigma$;
 7.          $\mathcal{D}'_2 = \mathcal{D}_2 \setminus \{\tau \in \mathcal{D}_2 \,|\, \mathrm{int}(\tau) \cap \Omega \neq \emptyset\}$;
 8.          form two cavities $C_1$, $C_2$ (in $|\mathcal{D}_2|$), where $C_1 \cap C_2 = \Omega$;
 9.          $\mathcal{D}_{C_1}$ = TetrahedralizeCavity($C_1$);
10.          $\mathcal{D}_{C_2}$ = TetrahedralizeCavity($C_2$);
11.          $\mathcal{D}_2 = \mathcal{D}'_2 \cup \mathcal{D}_{C_1} \cup \mathcal{D}_{C_2}$;
12.      endif
13.  endwhile
14.  return $\mathcal{D}_2$;
```

then updated to respect Ω with the new partitions of C_1 and C_2 (lines 11). Fig. 13 illustrates the idea of this algorithm in two dimensions.

6.2. Proof of Termination

For each missing region Ω, the FacetRecovery algorithm recovers it by modifying an *old tetrahedralization* $\mathcal{D}_2$ (which does not respect Ω) into a *new tetrahedralization* $\mathcal{D}_2$ which respects Ω, i.e., Ω is a union of faces in $\mathcal{D}_2$.

There are two key issues to be shown in the FacetRecovery algorithm, which are: *(1)* the two calls of the subroutine TetrahedralizeCavity (in lines 9 and 10), and *(2)* the update of $\mathcal{D}_2$ (line 11). For *(1)* we prove that each C_i, $i = \{1, 2\}$ can be partitioned into $\mathcal{D}_{C_i}$ without using Steiner points (Lemma 8). For *(2)*, we prove that the new tetrahedralization $\mathcal{D}_2$ (respects Ω) is a CDT with respect to all the recovered subfaces of $\mathcal{X}$ (Lemma 9).

Lemma 8. *Assume the old tetrahedralization $\mathcal{D}_2$ is a CDT and it satisfies the assumption 4. Then the two calls of* TetrahedralizeCavity *subroutines (in lines* 9 *and* 10*) success.*

Proof Recall that a cavity C in $|\mathcal{D}_2|$ is a polyhedron whose facets are all triangles. If a facet σ of C is a Delaunay in the vertex set $(\mathrm{vert}(C))$ of C, then the Delaunay tetrahedralization $\mathcal{D}_C$ of $\mathrm{vert}(C)$ must contain σ (by the assumption that $\mathrm{vert}(C)$ is in general position). If it is the case for every facet of C, then each tetrahedra in $\mathcal{D}_C$ must lie either inside or outside C. Hence the loop (lines $2-6$) in TetrahedralizeCavity can be done successfully. It remains to show that: all facets of C are Delaunay in $\mathrm{vert}(C)$. We show that this is true if the old tetrahedralization $\mathcal{D}_2$ (which does not respect Ω) is a CDT.

We divide the set of facets of C into two disjoint sets $\mathcal{K}_1$ and $\mathcal{K}_2$ such that all triangles in $\mathcal{K}_1$ are also triangles in $\mathcal{D}_2$, and no triangle of $\mathcal{K}_2$ is in $\mathcal{D}_2$. In other words, $\mathcal{K}_2$ is the set of missing subfaces in $\mathcal{D}_2$ and $|\mathcal{K}_2| = \Omega$ (see Fig. 14). Obviously, all facets of C in $\mathcal{K}_2$ are Delaunay in $\mathrm{vert}(C)$ (since each subface of $\mathcal{X}$ is constrained Delaunay in $\mathrm{vert}(\mathcal{X})$). What left is to show that: all facets in $\mathcal{K}_1$ are Delaunay in $\mathrm{vert}(C)$.

Assume the converse is true. Let $\sigma \in \mathcal{K}_1$ be a non-Delaunay face in $\mathrm{vert}(C)$. The diametric

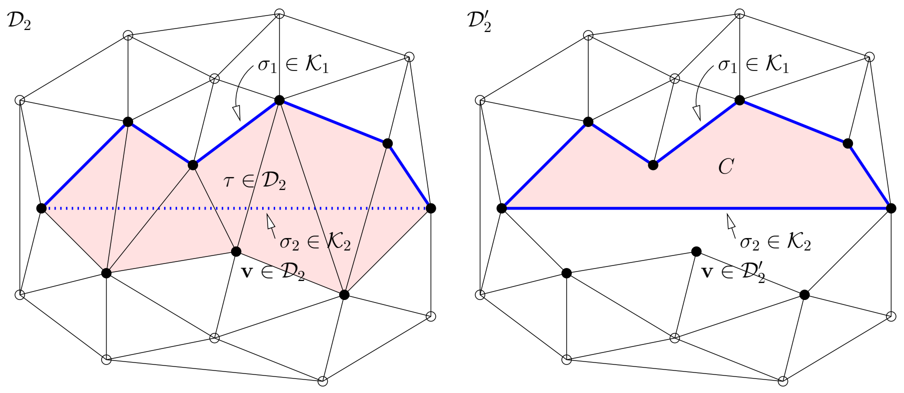

$\mathcal{D}_2$
$\sigma_1 \in \mathcal{K}_1$
$\tau \in \mathcal{D}_2$
$\sigma_2 \in \mathcal{K}_2$
$\mathbf{v} \in \mathcal{D}_2$
$\mathcal{D}_2'$
$\sigma_1 \in \mathcal{K}_1$
C
$\sigma_2 \in \mathcal{K}_2$
$\mathbf{v} \in \mathcal{D}_2'$

Figure 14. Proof of Lemma 8. Illustrations in 2D.

circumball B_σ of σ contains a subset V of vertices of vert(C) in its interior. Let H_σ be the plane passing through σ. H_σ divides V into two disjoint sets V_1 and V_2, where both V_1 and V_2 are not empty. Without loss of generality, let V_1 be the set of vertices at the same halfspace of H_σ containing the vertex $\mathbf{v} \in \mathcal{D}_2$, where $\mathbf{v}$ and σ form a tetrahedron τ in $\mathcal{D}_2$ ($\mathbf{v} \notin$ vert(C), see Fig. 14). Then all vertices of V_1 lie inside the circumscribed ball B_τ of τ.

Let $\mathbf{w} \in V_1$. We show that $\mathbf{w}$ is visible from the interior of τ. Pick any point $\mathbf{x} \in$ int(τ) such that the line segment $\mathbf{xw}$ intersects τ at a face ν of τ (such $\mathbf{x}$ exists). Since τ intersects Ω in its interior, it follows that ν also intersects Ω in its interior. If $\mathbf{w}$ is invisible from the interior of τ, then ν must belong to a facet of $\mathcal{X}$, so it can block the visibility of $\mathbf{w}$. However, this would implies that two facets of $\mathcal{X}$ intersect in their interior which is not possible by the definition of a PLS. So $\mathbf{w}$ must be visible from the interior of τ. Hence, $\tau \in \mathcal{D}_2$ is non-Delaunay, which implies that $\mathcal{D}_2$ is not a CDT. We arrive a contradiction. ■

Lemma 9. *The new tetrahedralization $\mathcal{D}_2$ (in line 11) is a CDT.*

Proof Let $\mathcal{D}_{old}$ denote the old tetrahedralization which does not respect Ω. By assumption, $\mathcal{D}_{old}$ is a CDT. $\mathcal{D}_2$ differs $\mathcal{D}_{old}$ only in the sets $\mathcal{D}_{C_1}$ and $\mathcal{D}_{C_2}$. It suffices to show that every simplex in $\mathcal{D}_{C_1}$ and $\mathcal{D}_{C_2}$ is constrained Delaunay in vert$(\mathcal{D}_2)$.

Without loss of generality, let σ be a 2-face of $\mathcal{D}_C$, where $\mathcal{D}_C$ may be either $\mathcal{D}_{C_1}$ or $\mathcal{D}_{C_2}$. We have the following cases.

(1) $\sigma \notin \partial\mathcal{D}_C$. Then σ is locally Delaunay (by TetrahedralizeCavity).
(2) $\sigma \in \partial\mathcal{D}_C$, and σ is a subface of $\mathcal{X}$. Then σ is locally Delaunay by definition.
(3) $\sigma \in \partial\mathcal{D}_C$, and σ is not a subface of $\mathcal{X}$. Then there are two tetrahedra $\tau \in \mathcal{D}_C$ and $\tau' \in \mathcal{D}_2 \setminus \mathcal{D}_C$ and $\sigma = \tau \cap \tau'$. Let $\mathbf{v} \in \tau$ and $\mathbf{v}' \in \tau'$ be the two opposite vertices of σ, respectively. If the circumscribed ball B_τ of τ contains $\mathbf{v}'$ in its interior, it also implies that $B_{\tau'}$ contains $\mathbf{v}$ in its interior. This implies that $\mathcal{D}_{old}$ is not a CDT since $\tau' \in \mathcal{D}_{old}$ is not Delaunay, a contradiction. Hence it must be that σ is locally Delaunay.

In all cases, σ is locally Delaunay. Then by Theorem 1, $\mathcal{D}_2$ is a CDT. ■

Now we can prove the termination of the FACETRECOVERY algorithm by combining Lemma 8 and Lemma 9.

Theorem 10. *The* FACETRECOVERY *algorithm terminates and results a CDT of $\mathcal{X}$.*

Proof At the beginning of the algorithm, the old tetrahedralization $\mathcal{D}_2 = \mathcal{D}_1$, it is a CDT. Hence the first missing region Ω_1 can be recovered (by Lemma 8) and the new tetrahedralization $\mathcal{D}_2$ (which respects Ω_1) is a CDT (by Lemma 9). This implies that the second missing region Ω_2 can be recovered in a new CDT $\mathcal{D}_2$ (which respects both Ω_1 and Ω_2). This process repeats iteratively. Since $\mathcal{X}$ has finite number of facets. It implies that there are finite number of missing regions. Hence the algorithm must terminate in a finite number of iterations. On finish, the resulting tetrahedralization $\mathcal{D}_2$ is a CDT of $\mathcal{X}$. ■

Theorem 10 completes the proof of the correctness of our CDT algorithm given in Section 4.

6.3. Complexity

In this section we show the worst behavior of the FACETRECOVERY algorithm with respect to the number of vertices and facets of the input PLS.

Lemma 11. *Let $\mathcal{X}$ be a three-dimensional PLS which has v vertices. One missing facet can be recovered in time $O(v^2 \log v)$.*

Proof We prove this lemma by constructing a PLS which needs such running time, then showing that it is indeed the worst case.

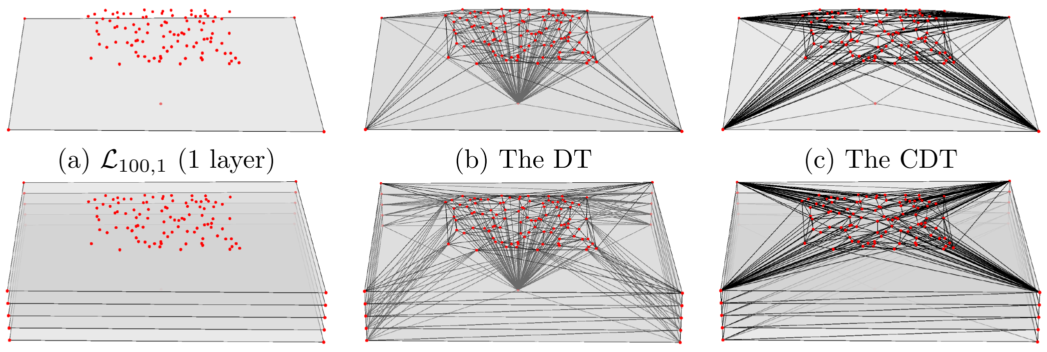
The PLS $\mathcal{L}_{v,1}$ (1-layer) shown in Fig. 15 (a) has only 1 facet. It is a slightly perturbed square (non-degenerate). $\mathcal{L}_{v,1}$ contains a set of v points (here $v = 100$) which are randomly distributed on top of the facet. The diameter of the facet is much larger than that of the point set. There is another point in $\mathcal{L}_{v,1}$ lies below the center of the facet. The Delaunay tetrahedralization of 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 [43], the worst time for constructing the Delaunay tetrahedralization is $O(v^2 \log v)$. Note that the largest possible size of a cavity is v. These together prove the claim. ■

Theorem 12. *Let $\mathcal{X}$ be a three-dimensional PLS which has v vertices and f facets. The* FACETRECOVERY *algorithm runs in time $O(fv^2 \log v)$.*

Proof Since the time for recovery of one facet may be $O(v^2 \log v)$, we just need to show that it is indeed possible that all facets of $\mathcal{X}$ may be missing and the recovery of any facet may require this time.

Let $\mathcal{L}_{v,f}$ be the PLS extended from $\mathcal{L}_{v,1}$ by including a set of f parallel facets. Fig. 15 (d) shows an example for $f = 5$. Clearly, the Delaunay tetrahedralization of 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



(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 15. 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 $\mathcal{L}_{100,5}$.

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 $O(fv^2 \log v)$. ■

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 $O(v^2 \log v)$, while the size of all other cavities is a constant (here is 4).

7. LOCAL DEGENERACIES REMOVAL

Theoretically, the assumption 4 can be satisfied by applying symbolic perturbation [46, 29, 47] on the geometric predicates, e.g., *point-in-sphere* test, during the CDT algorithm. If the input is a finite set of vertices, and the point-in-sphere test is calculated exactly, then the symbolic perturbation is enough to satisfy the assumption.

Note that the facets of a PLS can be defined by any number of coplanar vertices. Hence the point-in-sphere test is essentially reduced to the *point-in-circle* test on the vertices of a facet. However, the latter may not be calculated exactly due to the possible input error, e.g., the vertices defining a facet may not be exactly coplanar. Even the input data is correct, it may not be represented exactly by the computer floating-point numbers. For these reasons, a preprocessing on the input PLS is necessary. The purpose is to detect and break the (possible) degeneracies in the PLS. Steiner points (called *break point*) may be added.

Let $\mathcal{X}$ be a three-dimensional PLS. Let $F \in \mathcal{X}$ be a facet, and $\mathcal{K}_F$ be a triangulation of F. A set V of four vertices in F is called a *local degeneracy* if there are two triangles $\sigma, \nu \in \mathcal{K}_F$, where $\sigma \cap \nu \neq \emptyset$ and $V = \text{vert}(\sigma) \cup \text{vert}(\nu)$, such that the four vertices of V share a common sphere. See Fig. 16 left for examples of local degeneracies.

Once the triangulation of a facet is formed, the local degeneracies can be detected efficiently

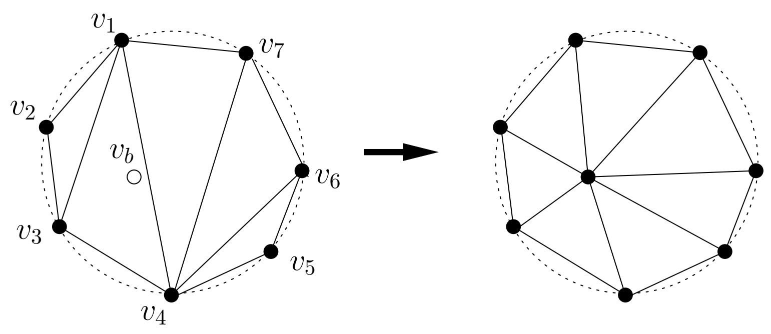


Figure 16. Local degeneracies and the break points. On the left, a set of seven coplanar vertices $\{\mathbf{v}_1, ..., \mathbf{v}_7\}$ sharing a common circle. A triangulation of the vertices is shown. The subset $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ is a local degeneracy, while the subset $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_5\}$ is not. There are 4 local degeneracies (corresponding to the four interior edges) in this triangulation. $\mathbf{v}_b$ is a break point. On the right is a triangulation after $\mathbf{v}_b$ is inserted, no local degeneracy exists.

by locally checking all adjacent pairs of triangles. Let V be a local degeneracy. Let $\Sigma(\mathbf{c}, r)$ be the common sphere shared by the four vertices of V, where $\mathbf{c}$ and r are the center and radius of Σ, respectively. We introduce a Steiner point $\mathbf{v}_b$, called *break point*, such that $\mathbf{v}_b = \mathbf{c} + \epsilon\,(\mathbf{c} - \mathbf{v})$, where $\mathbf{v} \in V$ and $0 < \epsilon < 1$. v_b locates inside Σ and will break the local degeneracy after it is inserted in to the triangulation by a Delaunay algorithm. See Fig. 16 right for an example.

The algorithm for removing local degeneracy is described below. The triangulation $\mathcal{F}$ of all facets of $\mathcal{X}$ is the input. The algorithm first initializes a queue Q containing all local degeneracies in $\mathcal{F}$. For each local degeneracy of $\mathcal{F}$, a break point $\mathbf{v}_b$ is calculated. v_b may encroach upon some segments. In such cases, $\mathbf{v}_b$ is shifted to the midpoint of an encroached segment. This avoids the case that $\mathbf{v}_b$ locates too close to a segment. Finally $\mathcal{F}$ is updated by including $\mathbf{v}_b$ as a vertex of it.

The success of this algorithm relies on the following hypothesis: the insertion of $\mathbf{v}_b$ does not create new local degeneracy in $\mathcal{F}$. If the parameter ϵ is chosen randomly for each $\mathbf{v}_b$, this hypothesis is true in high probability. If we ignore the time for updating $\mathcal{F}$ to include $\mathbf{v}_b$, this algorithm runs in linear time with respect to the initial size of Q.

8. BOUNDARY STEINER POINTS REMOVAL

In this section, we consider the *stronger 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 most 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 by no guarantee a CDT anymore. But this is not the question here.

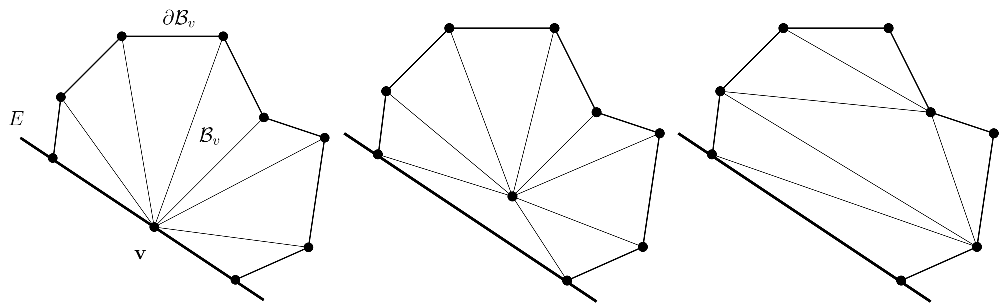


Figure 17. Boundary Steiner points removal (the 2D case). Left: $\mathbf{v}$ is a Steiner point on edge E. The half-ball $\mathcal{B}_v$ consists of all triangles having $\mathbf{v}$ as a vertex in one region. The bold lines are the boundary complex $\partial\mathcal{B}_v$. Middle: $\mathbf{v}$ is relocated into the interior of $\mathcal{B}_v$. Right: After $\mathbf{v}$ is deleted from $\mathcal{B}_v$.

8.1. 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 rather straightforward and practical. 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. 17 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. 17 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 it is able to relocate the Steiner point from the edge.

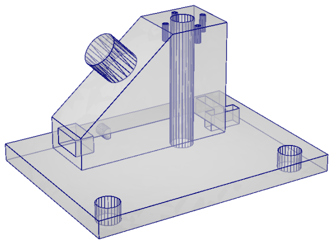
8.2. Point Deletion

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 [53] in which several criteria for selecting the contracting edges are proposed.

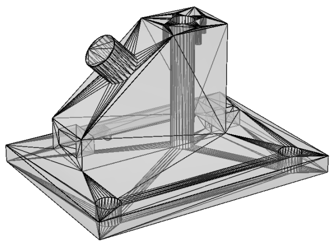
Another approach to directly delete $\mathbf{v}$ after forming the half-ball $\mathcal{B}_v$ is to re-tetrahedralize $\mathcal{B}_v$ such that the new tetrahedralization of $\mathcal{B}_v$ does not contain $\mathbf{v}$. A *flip-based approach* works in the following steps,

1. Re-triangulate the boundary of $\mathcal{B}_v$ such that $\mathbf{v}$ is not on $\partial\mathcal{B}_v$ any more.
2. Form a Delaunay tetrahedralization $\mathcal{D}$ of the vertices of $\mathcal{B}_v$ (without $\mathbf{v}$);
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 $\emptyset$.

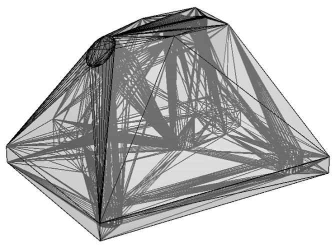
Although the above approach does not guarantee to work for all cases, it is shown in [54] that the flipping algorithm is effective in forming a tetrahedralization of $\mathcal{B}_v$. In Section 9, experiments on selected mesh examples are reported (see Table II).



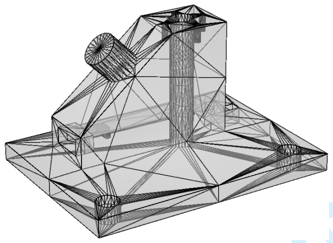
(a) A 3D PLS
460 vertices,
328 facets



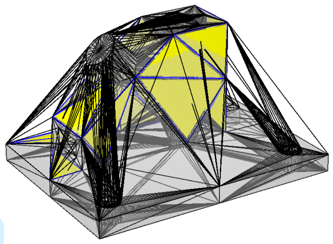
(b) The surface mesh
954 subfaces,
706 segments



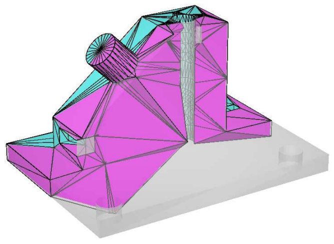
(c) The DT
460 vertices,
2637 tetrahedra



(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 18. Example: `Camila`. 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.

8.3. 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. [55, 56].

9. EXAMPLES AND DISCUSSIONS

The CDT algorithm has been implemented in the program `TetGen` [57]. Given a three-dimensional PLS, the LocalDegeneracyRemoval, SegmentRecovery, and FacetRecovery algorithms are called subsequently to create a CDT of that PLS. In the following, we provide several application examples to illustrate the practical behavior and the effectiveness of the CDT algorithm. Following the examples, we discuss the issues about the complexities of this algorithm and possible improvements.

Fig. 18 illustrates an example of one run of the CDT algorithm on a mechanical part (`Camila`, available from [58]) with the intermediate status of the different steps. The input PLS shown in (a) has 460 vertices, 706 segments, and 328 facets. The surface mesh shown in (b),

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.)

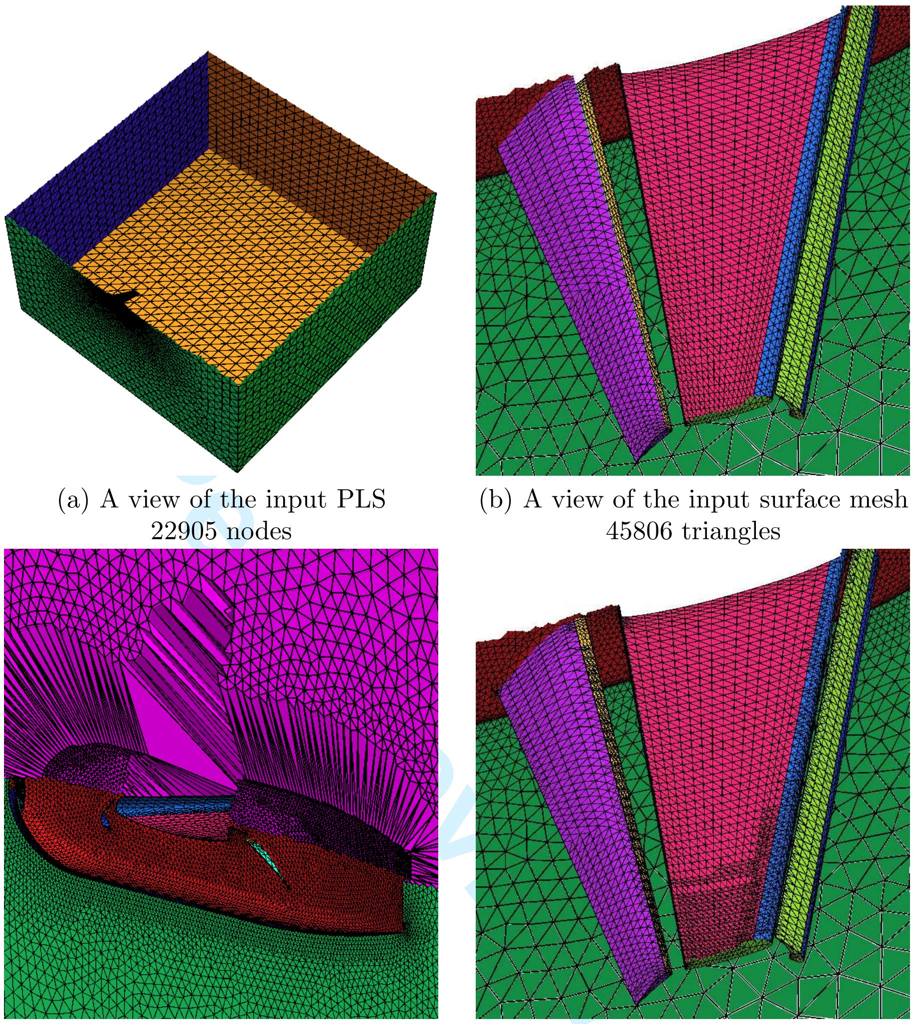
which is the input of the local degeneracy removal algorithm, contains 954 subfaces. (c) is the initial Delaunay tetrahedralization of the vertex set. The status after the SegmentRecovery algorithms is shown in (d). The number of break points and protect points are 213 and 269, respectively. (e) shows the initial status of the FacetRecovery algorithm, there are total 6446 subfaces in which 22 are missing (highlighted in yellow). The resulting CDT is shown in (f). A vertical cut is made for visualizing the interior constrained Delaunay tetrahedra.

In the above example, the number of Steiner points is in the same order of the input size. This is the most often case that we have observed. Note that it may be further reduced the number of Steiner points by using appropriately point insertion order (currently it is random). Also note that the local degeneracy removal algorithm may be called only when it is necessary. Current we call it in advance which may add at most $O(n)$ Steiner points.

The geometry of the next example shown in Fig. 19 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 total 19,455 Steiner points in which 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 running 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 [58]. Two of the generated CDTs are shown in Fig. 20. 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 running 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 time comparing to other steps. While the times for facet recovery (line 11) were relatively small which mean there were



(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 19. 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).

relatively small number of missing facets. The step of local degeneracy removal (line 9) uses linear time with respect to the input number of facets (line 4).

Our CDT algorithm will inserted Steiner points on the boundary (segments and facets) of the input PLS. In Section 8 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 few Steiner points remaining inside the PLS. This step is implemented in `TetGen`.

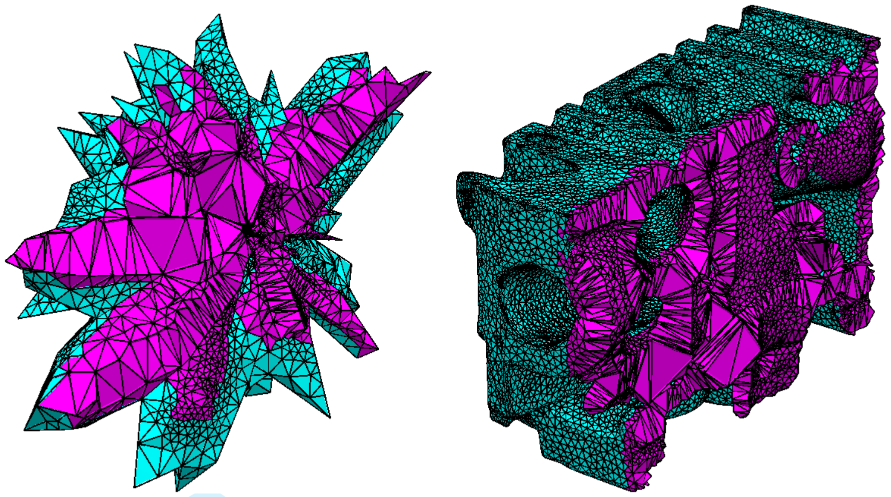


Figure 20. Examples: `Crystal` (left) and `IFP` (right). The generated CDTs are shown. See Table I for their statistics.

	`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. Experiments on the post-process step for Steiner point removal (discussed in Section 8). Tested by `TetGen` (the same version in Table I).

Table II reports the experiments of this step on some selected examples. From Table II, we can observe that the majority of these Steiner points could be completely deleted from the mesh, only few of them are relocated in the interior of the domain.

Although our algorithm does not directly work on smooth domains (whose boundaries are curved surfaces), it is possible to combine our algorithm with other surface meshing algorithms, e.g., [59, 60], 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 *restricted Delaunay triangulation* of a set of ϵ-samples on Σ [61, 62]. Fig. 21 left shows a restricted Delaunay triangulation of a piecewise smooth surface. This model is freely available from [63]. Since such triangulation is a two-dimensional PLS. Hence the domain bounded by it can be approximated by a CDT, see Fig. 21 left. In particular, restricted Delaunay triangulations are subsets of Delaunay triangulations. Theoretically, our CDT algorithm needs no Steiner points in constructing the CDTs from such inputs.

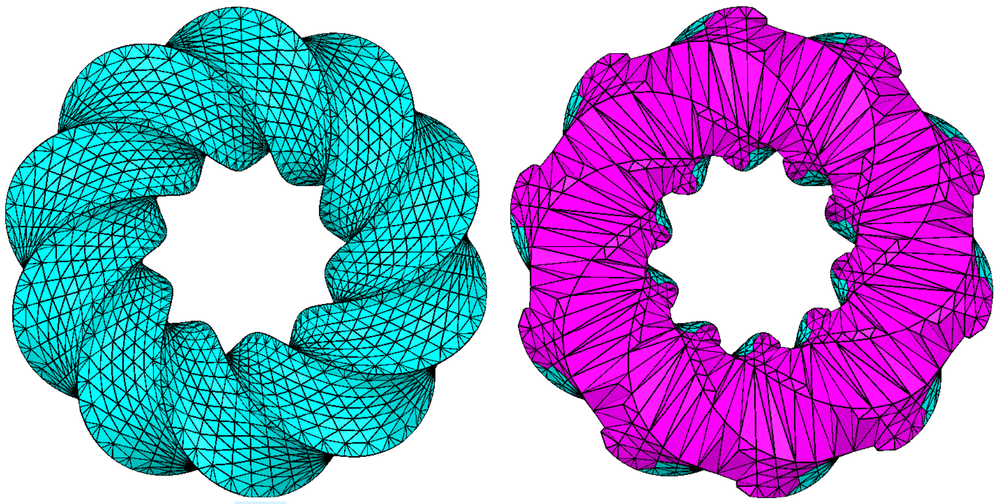


Figure 21. Examples `n10`. Left: a restricted Delaunay triangulation of a piecewise smooth surface. Right: a CDT partition of the domain bounded by this surface (a cut away view of the interior tetrahedra), it contains no Steiner points.

10. CONCLUSION

The three-dimensional boundary recovery problem has challenges both in theory and practice. We showed that constrained Delaunay tetrahedralizations (CDTs) are useful in solving this problem. An algorithm to construct a CDT of an arbitrary three-dimensional piecewise linear system (PLS) is developed and its correctness is proved. The essential steps of this algorithm are simple and easy to implement. The choices of Steiner points are determined through three segment splitting rules and the recovery of facets is by constructing locally Delaunay tetrahedralizations. We provided an analysis on this algorithm. In the following, we summarize the results of this paper and outline some problems for the future work.

- We defined 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. It is more general than previous definitions. We showed several basic properties of such objects which are very 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. The termination and correctness of this algorithm are proved. We give partial analysis on 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 [29, 64, 65] 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 are not forced to be split in advance.

For future work, some theoretical questions regarding this algorithm are worth to 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 [33] in computational geometry, i.e., the upper bound of the number of Steiner points needed to construct a conforming Delaunay tetrahedralization of a three-dimensioanl PLS?
- Limiting the number of Steiner points is important since it directly influences the performance of the facet recovery algorithm. Some technical detail of the segment recovery algorithm need to be investigated. 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 boundary is discussed and implemented, the problem is far from been solved. Difficulties are arising in practices when the input boundary of the PLS contains anisotropic features.
- The proposed CDT algorithm only takes 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* [60] could be considered.

ACKNOWLEDGEMENTS

This work was supported by the WIAS research project `Numerical Mthods` in the topic of `numerical solution of partial differential equations`. The authors would like to thank Jürgen Fuhrmann for stimulating of this work and many helpful discussions. We thank the referees for their valuable comments and suggestions, and the `GAMMA` project from INRIA for open access to their mesh repository. Special thanks to Larry Wigton from the Boeing company for providing the `Wing-Iso` model.

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. Du Q, Wang D. Constrained boundary recovery for the three dimensional Delaunay triangulations. *International Journal for Numerical Methods in Engineering* 2004; **61**:1471–1500.
9. Chazelle B, Palios L. Triangulating a nonconvex polytope. *Discrete and Computational Geometry* 1990; **5**:505–526.
10. Dey TK. Triangulating and csg representation of polyhedra with arbitrary genus. *Proc. 7th annual ACM Symposium on Computational Geometry*, 1991; 364–372.

11. Bern M. Compatible tetrahedralizations. *Proc. 9th annual ACM Symposium on Computational Geometry*, 1993; 281–288.
12. Chazelle B, Shouraboura N. Bounds on the size of tetrahedralizations. *Discrete and Computational Geometry* 1995; **14**:429–444.
13. Joe B. Tetrahedral mesh generation in polyhedral regions based on convex polyhedron decompositions. *International Journal for Numerical Methods in Engineering* 1994; **37**:693–713.
14. Hershberger JE, Snoeyink J. Erased arrangements of lines and convex decompositions of polyhedra. *Computational Geometry* 1998; **9**(3):129–143.
15. Edelsbrunner H. *Algorithms in combinatorial geometry*. Springer-Verlag: Heidelberg, 1987.
16. De Berg M, Van Kreveld M, Overmars M, Schwarzkopf O. *Computational Geometry: Algorithms and Applications*. Springer-Verlag: Heidelberg, 1997.
17. Delaunay BN. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 1934; **7**:793–800.
18. Pébay P, Frey PJ. A priori Delaunay-conformity. *Proc. 7th International Meshing Roundtable*, Sandia National Laboratories, 1998; 321–333.
19. 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.
20. Cohen-Steiner D, De Verdière EC, Yvinec M. Conforming Delaunay triangulation in 3D. *Proc. 18th annual ACM Symposium on Computational Geometry*, 2002.
21. Edelsbrunner H, Tan TS. An upper bound for conforming Delaunay triangulations. *SIAM Journal on Computing* 1993; **22**:527–551.
22. Lee DT, Lin AK. Generalized Delaunay triangulations for planar graphs. *Discrete and Computational Geometry* 1986; **1**:201–217.
23. Chew PL. Guaranteed-quality triangular meshes. *Technical Report TR 89-983*, Department of Computer Science, Cornell University 1989.
24. Shewchuk JR. General-dimensional constrained Delaunay and constrained regular triangulations I: Combinatorial properties 2007. To appear in Discrete and Computational Geometry.
25. Baker T. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers* 1989; **5**:161–175.
26. Hazlewood C. Approximating constrained tetrahedralizations. *Computer Aided Geometric Design* 1993; **10**:67–87.
27. Cavalcanti PR, Mello UT. Three-dimensional constrained Delaunay triangulation: a minimalist approach. *Proc. 8th International Meshing Roundtable*, Sandia National Laboratories, 1999; 119–129.
28. Si H. Adaptive tetrahedral mesh generation by constrained delaunay refinement. *International Journal for Numerical Methods in Engineering* 2008; **75**(7):856–880.
29. Shewchuk JR. Constrained Delaunay tetrahedralization and provably good boundary recovery. *Proc. 11th International Meshing Roundtable*, Sandia National Laboratories, 2002; 193–204.
30. Shewchuk JR. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. *Proc. 19th Annual Symposium on Computational Geometry*, 2003; 181–190.
31. Si H, Gärtner K. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. *Proc. 14th International Meshing Roundtable*, Sandia National Laboratories: San Diego, CA, USA, 2005; 147–163.
32. Shewchuk JR. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. *Proc. 14th Annual Symposium on Computational Geometry*, 1998; 76–85.
33. Edelsbrunner H. *Geometry and topology for mesh generation*. Cambridge University Press: England, 2001.
34. 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.
35. Si H. Three dimensional boundary conforming Delaunay mesh generation. PhD Thesis, Faculty II - Mathematics and Natural Sciences, Technische Universität Berlin 2008.
36. Huang W. Measuring mesh qualities and application to variational mesh adaptation. *SIAM Journal on Scientific Computing* 2005; **26**(5):1643–1666.
37. D'Azevedo EF, Simpson RB. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing* 1989; **6**:1063–1075.
38. Chen L, Xu JC. Optimal Delaunay triangulations. *Journal of Computational Mathematics* 2004; **22**(2):299–308.
39. Rajan VT. Optimality of the Delaunay triangulation in $\mathbb{R}^d$. *Discrete and Computational Geometry* 1994; **12**:189–202.
40. Clarkson KL, Shor PW. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry* 1989; **4**:387–421.
41. Bowyer A. Computing Dirichlet tessellations. *Comp. Journal* 1987; **24**(2):162–166.

42. Watson DF. Computing the n-dimensional Delaunay tessellations with application to Voronoi polytopes. *Comput. Journal* 1987; **24**(2):167–172.
43. Edelsbrunner H, Shah NR. Incremental topological flipping works for regular triangulations. *Algorithmica* 1996; **15**:223–241.
44. 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.
45. Chew PL. Constrained Delaunay triangulation. *Algorithmica* 1989; **4**:97–108.
46. 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.
47. Devillers O, Teillaud M. Perturbations and vertex removal in Delaunay and regular 3D triangulations. *Technical Report 5968*, INRIA 2006.
48. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms* 1995; **18**(3):548–585.
49. Mücke EP, Saias I, Zhu B. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Proc. 12th annual Symposium on Computational Geometry*, 1996; 274–283.
50. Shewchuk JR. Stabbing delaunay tetrahedralizations. *Discrete and Computational Geometry* 2004; **32**:339–343.
51. CGAL. Computational Geometry Algorithms Library. `http://www.cgal.org` 2007.
52. Shewchuk JR. Robust adaptive floating-point geometric predicates. *Proc. 12th annual Symposium on Computational Geometry*, 1996; 141–150.
53. Ollivier-Gooch CF. Coarsening unstructured meshes by edge contraction. *International Journal for Numerical Methods in Engineering* 2003; **57**:391–414.
54. 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.
55. Freitag L, Ollivier-Gooch CF. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering* 1997; **40**(21):3979–4002.
56. Klingner BM, Shewchuk JR. Aggressive tetrahedral mesh improvement. *Proc. 16th International Meshing Roundtable*, Sandia National Laboratories, 2007; 3–23.
57. Si H. TetGen. `http://tetgen.berlios.de` 2007.
58. INRIA. GAMMA, automatic mesh generation and adaption methods. `http://www-c.inria.fr/gamma` 2007.
59. Boissonnat JD, Oudout S. Provably good surface sampling and meshing of surfaces. *Graphical Models* 2005; **67**:405–451.
60. Cheng SW, Dey TK, Ramos EA. Delaunay refinement for piecewise smooth complexes. *Proc. 18th annual ACM-SIAM Symposium on Discrete Algorithms*, 2007; 1096–1105.
61. Amenta N, Bern M. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry* 1999; **22**:481–504.
62. Dey T. *Curve and Surface Reconstruction : Algorithms with Mathematical Analysis*. Cambridge University Press: England, 2006.
63. Edelsbrunner H. 180 wrapped tubes. `http://www.cs.duke.edu/ edels/Tubes`.
64. 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.
65. Pav SE, Walkington NJ. Robust three dimensional Delaunay refinement. *Proc. 13th International Meshing Roundtable*, Sandia National Laboratories, 2004.