Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations

Hang Si and Klaus Gärtner

Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany
{si, gaertner}@wias-berlin.de

Summary. We present a method to decompose an arbitrary 3D piecewise linear complex (PLC) into a constrained Delaunay tetrahedralization (CDT). It successfully resolves the problem of non-existence of a CDT by updating the input PLC into another PLC which is topologically and geometrically equivalent to the original one and does have a CDT. Based on a strong CDT existence condition, the redefinition is done by a segment splitting and vertex perturbation. Once the CDT exists, a practically fast cavity retetrahedralization algorithm recovers the missing facets. This method has been implemented and tested through various examples. In practice, it behaves rather robust and efficient for relatively complicated 3D domains.

1 Introduction

A fundamental problem in unstructured mesh generation is to create a mesh that represents a geometric domain bounded by piecewise linear faces and possibly with interior constraining faces. It is also referred as boundary mesh generation. Numerous applications depend on it.

This problem has been successfully solved in two dimensions. It is well known that every polygonal domain can be triangulated into triangles without adding new vertices (the Steiner points). Almost optimal algorithms (with a linear complexity in practice) have been proposed [Lee86, Chew89].

The problem is significantly more difficult for arbitrarily shaped three dimensional domains. Such domains can be described by *piecewise linear complexes* (PLCs) [Miller96], which are objects more general than polyhedra. It is known [Schoenhardt28, Bagemihl48, Chazelle84, Rambau03] even a simple polyhedron may not be tetrahedralizable without adding new vertices. The simplest example is the so-called Schönhardt polyhedron [Schoenhardt28], which is a non-convex twisted triangular prism. Moreover, the problem of deciding whether a simple polyhedron can be tetrahedralized is NP-hard [Ruppert92]. PLCs are usually much more complicated than simple polyhedra. To guarantee an arbitrary PLC can always be meshed, methods must resort to adding Steiner points. However, a number of difficult issues remain to be resolved, such as the placement of the Steiner points, the minimum bound on such points, and so on.

The recently proposed *Constrained Delaunay tetrahedralization* (CDT) (cf. [Shewchuk02]) is a Delaunay-like tetrahedralization that is constrained to respect the shape of a PLC. CDTs are obviously suitable structures for resolving the above problem. Not only they respect the boundary but also they have many nice mathematical properties inherited from Delaunay tetrahedralizations. Many applications can be envisaged after getting a CDT. For instance, it is a good initial mesh for getting a quality conforming Delaunay mesh [Shewchuk98b, Cheng04, Pav04] which is suitable for numerical methods.

A key question for constructing a CDT is to decide its existence, i.e., whether a given PLC has a CDT without adding points. So far, Shewchuk [Shewchuk98a] has proved the condition: if all segments of the PLC are *strongly Delaunay*, then the CDT exists. The hint gained from this condition is: additional points can be inserted only on the segments of the PLC.

Shewchuk [Shewchuk02] gave a segment recovery algorithm for constructing CDTs. For a PLC X, it carefully introduces additional points on some segments of X until the existence of a CDT can be guaranteed. This algorithm yields a provably good bound on edge lengths. However, it requires to compute the local feature size explicitly for protecting *sharp corners* (vertices with angles less than 90° formed by segments). In [Si04a] we proposed a new segment recovery strategy which exploits the available local geometric information to efficiently construct Steiner points on segments. It needs not to compute the local feature sizes. Moreover sharp corners are implicitly handled during the creation of the Steiner points. Both algorithms tend to use fewer additional points than other methods which do edge protect provably [Pebay98, Murphy00, Cohen-Steiner02] too.

When the existence of a CDT is known, another key issue is to recover facets of the PLC. Generally non-CDT algorithms [Pebay98, Murphy00, Cohen-Steiner02, Weatherill94, George03, Du04] will continuously insert points on the missing facets or the inside of the PLC. While CDT algorithms [Shewchuk02, Shewchuk03, Si04a] recover the missing facets without introducing additional points. This again reduces the number of Steiner points in a CDT.

By now Shewchuk has provided several facet recovery algorithms [Shewchuk00, Shewchuk02, Shewchuk03]. The incremental facet insertion algorithm [Shewchuk02] recovers facets one after one and the CDT is updated accordingly. For each facet, a gift-wrapping algorithm is used for retetrahedralizing the two cavities around it. The strategy is simple but the time complexity is poor and the performance is unstable due to the gift-wrapping algorithm. The flip-based algorithm [Shewchuk03] recovers each facet by a sequence of carefully ordered flip operations. It appears to be simple and is likely to outperform other algorithms on most inputs. In order to guarantee the correctness and termination, both algorithms require that a full perturbation has to be applied on the set of vertices to remove the degeneracies.

In this paper we present a new CDT algorithm. The main difference from other CDT algorithms is the practical exploitability of a strong CDT existence condition which requires no *local degeneracy* on the vertices of the PLC. We propose a local degeneracy removal algorithm to construct a new set of vertices out of the old one which is consistent with the constraining segments and facets of the PLC. After the strong condition is satisfied, facet recovery is done by a new cavity retetrahedralization algorithm which is fast and robust in practice.

The remainder of this paper is organized as follows. We shortly recall the definitions of PLCs and CDTs in the next section. Then the existence of CDT is discussed

in section 3. Section 4 provides an overview of the proposed method. The individual algorithms are fully described in section 5, 6, and 7. Finally we present some meshing results from publicly available examples.

2 PLCs and CDTs

A *piecewise linear complexes* (PLC) proposed by Miller, Teng, Walkington, and Wang [Miller96] is a general boundary description for three-dimensional domains. Simply saying, a PLC X is a set of vertices, together with a collection of segments, and facets. Like a simplical complex, any two components of X are either disjoint or meet in a common face, e.g., two segments can only intersect at a vertex of X, and two facets can only intersect at a collection of segments and vertices of X, and so on. A PLC facet has no analogue in a simplical complex. Each facet of X is indeed a two-dimensional polygonal region embedded in three dimensions, it may not be convex and possibly contains holes, isolated segments and vertices.

PLCs are more general than polyhedra in the sense that every polyhedron is a PLC but not vice versa. For instance, a PLC containing a segment inside can't be represented by any polyhedron. Surface triangulations are one special type of PLCs - each facet is a triangle. Hence PLCs are able to approximate arbitrary complicated and curved shapes. In addition, many popular polygonal file formats (e.g., STL, OFF, PLY, and [Si04b]) can be directly used or slightly modified to describe PLCs.

Given a PLC X, Shewchuk [Shewchuk02] defined a CDT of X as follows:

Let V be the set of vertices of X. σ is any simplex (tetrahedron, triangle, edge or vertex) formed by vertices of V. σ is *Delaunay* if there exists a circumsphere of σ that encloses no vertex of V. The *Delaunay tetrahedralization* $\mathcal{D}$ of V is a tetrahedralization that all simplices of $\mathcal{D}$ are Delaunay. $\mathcal{D}$ is unique if V is general, i.e., no five or more vertices lie on a common sphere.

The visibility between two vertices p and q is *occluded* if there is a constraining facet f such that p and q lie on opposite sides of the plane that includes f, and the line segment pq intersects this facet(see Figure 1). Segments do not occlude visibility. For example, in Figure 1, c and d can see each other even if ab is a segment.

A tetrahedron t formed by vertices of X is *constrained Delaunay* if its circumsphere encloses no vertex of X which is visible from any point in the relative interior of t (see Figure 1).

A tetrahedralization $\mathcal{T}$ is a *constrained tetrahedralization* of X if $\mathcal{T}$ and X have exactly the same vertices, and the tetrahedra in $\mathcal{T}$ cover the convex hull of V. Every segment of X is an edge of $\mathcal{T}$, every facet of X is a union of triangular faces of $\mathcal{T}$.

A constrained tetrahedralization $\mathcal{T}$ of X is said to be a *constrained Delaunay tetrahedralization* (CDT) of X if each tetrahedron of $\mathcal{T}$ is constrained Delaunay.

Intuitively, the definitions of Delaunay tetrahedralization and constrained Delaunay tetrahedralization are the same except that, for the CDT, we ignore the volume of a sphere whenever the sphere passes through a facet of X.

Let $\mathcal{T}$ be a CDT of X. A facet of X is represented by a set of coplanar triangular faces of $\mathcal{T}$. Such faces are called *subfaces* for distinguishing them from other faces of $\mathcal{T}$. The set of subfaces of a facet form a two-dimensional constrained Delaunay triangulation.

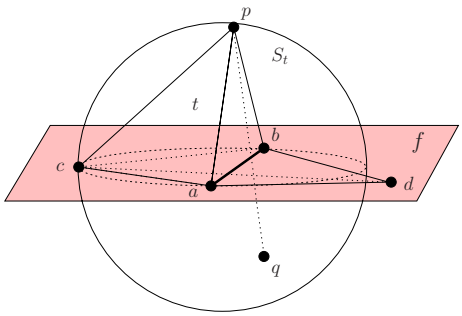

Fig. 1. Constrained Delaunay tetrahedron. The shaded region represents a facet f of X including vertices a, b, c and d. Vertices p and q lie on opposite sides of f, they can not see each other. While c and d can see each other even if ab is a segment of X. S_t is the circumsphere of tetrahedron t ($abcp$) and it encloses q but not d, t is constrained Delaunay.

3 The Existence of CDT

Given a PLC X. Generally, the CDT of X may not exist. One reason is that X may not be tetrahedralizable at all. Even the constrained tetrahedralization of X exists, it may still not have a CDT. A key question for CDT algorithms is to decide under what condition the CDT exists.

Shewchuk has proved a condition. Let σ be a simplex of X, σ is *strongly Delaunay* if there exists a circumsphere of σ that passes through and encloses no other vertices of X. Say X is *edge-protected* if every segment of X is strongly Delaunay.

Theorem 1 ([Shewchuk98a]). *If X is edge-protected, then X has a CDT.* ■

This condition is useful in practice because it suggests that additional points can be inserted on segments only. However, this condition does not help us to construct the CDT. Suppose X has been made edge-protected (by splitting the segments into smaller segments), algorithms [Shewchuk02, Shewchuk03, Si04a] need to carefully do perturbations on the vertices of X in order to successfully recover the missing subfaces.

In this paper, we use another less general condition which guarantees the existence of CDT, too.

Definition Let $\mathcal{D}$ be the Delaunay tetrahedralization of the vertices of X, t a tetrahedron in $\mathcal{D}$ and t' an adjacent tetrahedron of t (sharing a face with t), V the set of vertices of t, t'. If all vertices of V lie on a common sphere, V is called a *local degeneracy* of X.

Remark: If X contains no local degeneracy, then $\mathcal{D}$ is unique. However, the set of vertices of X may still be degenerate (since arbitrary 5 or more vertices of X can lie on a common sphere).

Theorem 2. *If $\mathcal{D}$ contains no local degeneracy and contains all segments of X, then the CDT of X exists.*

Proof: It is easy to show: if X satisfies the above condition, then the segments of X are edge-protected. ■

This condition is stronger than Shewchuk's condition since it implicitly satisfies it. Its advantage is: the facet recovery can be carried out efficiently. Section 7 presents such an algorithm.

4 The Algorithm

Let the initial PLC be X_0. The CDT is constructed by the following consecutive steps:

(1) Construct an initial Delaunay tetrahedralization $\mathcal{D}_0$ of the vertices of X_0.
(2) Recover the segments of X_0 in $\mathcal{D}_0$ by incrementally inserting points on missing segments, update $X_0 \rightarrow X_1$ and $\mathcal{D}_0 \rightarrow \mathcal{D}_1$ with the newly inserted points respectively.
(3) Remove the local degeneracies in X_1 by either perturbing vertices or inserting new vertices, update $X_1 \rightarrow X_2$ and $\mathcal{D}_1 \rightarrow \mathcal{D}_2$ with the newly inserted points respectively.
(4) Recover the subfaces of X_2 in $\mathcal{D}_2$ by a cavity retetrahedralization method.

In step (1), $\mathcal{D}_0$ can be efficiently constructed by any standard algorithm, such as [Edelsbrunner96]. $\mathcal{D}_0$ probably does not respect the segments and subfaces of X_0. After step (2) is done, D_1 is a Delaunay tetrahedralization of vertices of X_1 which contains all segments of X_1. However, D_1 may not respect the subfaces of X_1. Step (3) guarantees the existence of a CDT. After all local degeneracies are removed, $\mathcal{D}_2$ is the unique Delaunay tetrahedralization of vertices of X_2 and contains all segments of X_2. D_2 may still not respect the subfaces of X_2, while the existence of the CDT (of X_2) is guaranteed. Hence the step (4) can be carried out without inserting additional vertices. These processes are detailed in the following sections.

5 Segment Recovery

A segment of X_0 is *missing* if it is not in $\mathcal{D}_0$. The purpose of the segment recovery algorithm is to update $\mathcal{D}_0$ into $\mathcal{D}_1$ such that $\mathcal{D}_1$ is a Delaunay tetrahedralization and includes all segments of X_0.

Let e_ie_j be a segment with endpoints e_i and e_j, $|e_ie_j|$ be its Euclidean length. A vertex is *acute* if at least two segments incident at it form an angle smaller than 90°. We distinguish two types of segments, a segment is *type-1* if its both endpoints are not acute, it is *type-2* if only one of its endpoints is acute. If both endpoints of a segment are acute, it can be transformed into two type-2 segments by inserting a vertex.

A segment is split into *subsegments*. Subsegments inherit types from original segments. For example, let e_ie_j be a subsegment of e_1e_2 which is a type-2 segment and e_1 is acute, e_ie_j is type-2 although none of its endpoints is acute. For any vertex v inserted on a type-2 segment (or subsegment), $O(v)$ denotes its original acute vertex. A tacit rule is used throughout this section, if e_ie_j is type-2, it implies either

e_i or $O(e_i)$ is the acute vertex. In the following, unless it is explicitly mentioned, a segment can be a segment or subsegment.

A vertex *encroaches upon* a segment if it lies inside or on the diameter sphere of that segment. *Remark*: any missing segment must be encroached by at least one vertex of X.

Let e_ie_j be a missing segment, it will be split by a vertex v. The *reference point* p of v, which is "responsible" for the insertion of v, is defined as follows:

- p encroaches upon e_ie_j;
- the circumradius of the smallest circumsphere of triangle e_ie_jp is maximum over other encroaching points of e_ie_j.

Figure 2 illustrates how p is chosen. Notice that p may not unique (because several points can share the same circumsphere), choose an arbitrary one in this case.

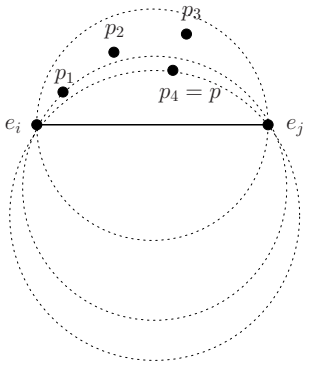

Fig. 2. The reference point p of a splitting segment e_ie_j. p_1, p_2, p_3 and p_4 all encroach upon e_ie_j. p_4 is chosen as the reference point because it forms the biggest circumsphere with e_ie_j.

The insertion of v to split e_ie_j is governed by three rules given below. Let $S(c, r)$ denote a sphere centered at c with radius r:

1. e_ie_j is type-1 (see Figure 3 (a)), then $v := e_ie_j \cap S$, where $S(c, r)$ is the sphere defined by the reference point p of v as follows:
   if $|e_ip| < \frac{1}{2}|e_ie_j|$ **then**
   $c := e_i, r := |e_ip|$,
   else if $|e_jp| < \frac{1}{2}|e_ie_j|$ **then**
   $c := e_j, r := |e_jp|$,
   else
   $c := e_i, r := \frac{1}{2}|e_ie_j|$,
   end
2. e_ie_j is type-2 (see Figure 3 (b)), let $e_k := O(e_i)$, then $v := e_ke_j \cap S$, where $S(c, r)$ is the sphere defined by the reference point p of v with $c := e_k, r := |e_kp|$. However, if the subsegment ve_j has length $|ve_j| < |vp|$, then we do not insert v and use rule 3 to split the segment.

3. e_ie_j is type-2, let $e_k := O(e_i)$, and v' is the rejected vertex by rule 2. then $v := e_ke_j \cap S$, where $S(c, r)$ is the sphere defined by the reference point p of v as follows (see Figure 3 (c)):
   $c := e_k$
   if $|pv'| < \frac{1}{2}|e_iv'|$ **then**
   $r := |e_ke_i| + |e_iv'| - |pv'|$
   else
   $r := |e_ke_i| + \frac{1}{2}|e_iv'|$
   end

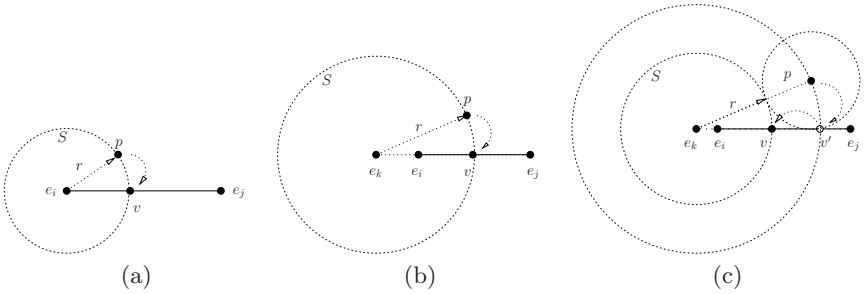

Fig. 3. Illustrations of the segment splitting rules.

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 Figure 4. Notice, the protecting ball is not necessarily completely created, only the missing segments will be split and protected. Existing segments remain untouched. This reduces the number of Steiner points.

Below is the pseudo-code of the segment recovery algorithm.

Algorithm *Delaunay Segments Recovery*
Input: $\mathcal{D}_0$, X_0.
Output: $\mathcal{D}_1$, X_1.
initialize:
  $\mathcal{D}_1 := \mathcal{D}_0$, $X_1 := X_0$;
repeat:
  form a queue Q of missing segments in $\mathcal{D}_1$;
  while $Q \neq \emptyset$ **do**
    remove a segment e_ie_j from Q;
    split e_ie_j using rule 1, or 2, or 3;
    update $\mathcal{D}_1$, X_1;
  end
until no segment of X_1 is missing in $\mathcal{D}_1$

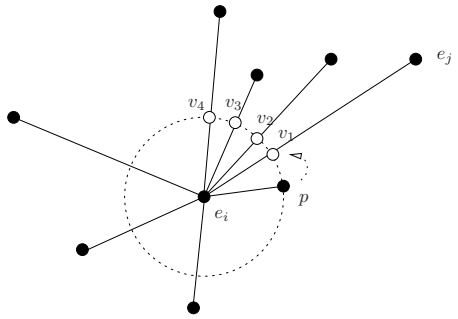

Fig. 4. The protecting ball of an acute vertex e_i. v_1, v_2, v_3, and, v_4 are points inserted on segments (by rule 2) sharing e_i. They automatically create a protecting ball of e_i.

The termination of this algorithm can be proved by showing that the length of every segment created is bounded by the local feature size divided by constant depending only on the input. For a PLC X, the *local feature size* [Ruppert95] $lfs(v)$ of any point v in X is the radius of the smallest ball centered at v that intersects two segments or vertices in X that do not intersect each other. The $lfs()$ defines a continuous map that maps every point in X into a positive value which suggests how large the ball of the empty space around this point can be. The function $lfs()$ is only defined on the input PLC X and does not change as new points are inserted.

Theorem 3 ([Si04a]). *Let e_ie_j be a finally resulting subsegment,*

- *if e_ie_j is type-1, then:*
  $|e_ie_j| \geq min\{lfs(e_i), lfs(e_j)\}$.
- *if e_ie_j is type-2, let $e_k := O(e_i)$, then:*
  $|e_ie_j| \geq \frac{1}{C} lfs(e_k)$ *when* $e_i = e_k$,
  $|e_ie_j| \geq lfs(e_k)sin(\theta)$ *when* $e_i \neq e_k$.
  where C is two times the number of segments incident at e_k and θ is the smallest angle between them.

hence the Delaunay segments recovery algorithm terminates. ■

Practically, the algorithm terminates within a few steps creating the protection ball sector. The constant C usually is no larger than 4.

6 Removing Local Degeneracies

In order to fulfill the condition of Theorem 2, all local degeneracies have to be removed from $\mathcal{D}_1$. Techniques of perturbation [Edelsbrunner90] are effective to remove degeneracies. However, they must be carefully applied in CDT algorithms.

We say a vertex of a PLC is *perturbable* if there exists an arbitrarily small perturbation on it which does not affect the consistency of the PLC, otherwise, it is

unperturbable. For example, a vertex on a segment or inside a facet is perturbable since it can be perturbed arbitrarily along the segment vector or inside the facet without affecting the collinearity of the segment or the coplanarity of the facet. Not every vertex of a PLC is simply perturbable. If a vertex intersected by three or more non-coplanar facets is perturbed, at least one facet becomes invalid (because it now contains a non-coplanar vertex), hence it is unperturbable.

Let Δ be a local degenerate set of vertices in X_1, i.e., Δ contains 5 vertices of X_1 which share a common sphere. If $p \in \Delta$ is perturbable, Δ can be removed by an arbitrarily small perturbation on p. We call a perturbation is *segment-safe* if after the perturbation no segment of X_1 becomes non-Delaunay. A perturbable vertex may not be segment-safe, see Figure 5 for an example.

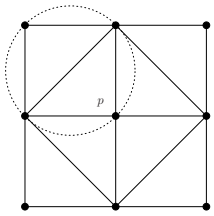

Fig. 5. A perturbable vertex which is not segment-safe. The vertices of this facet consist of a 3×3 square grid. The vertex p is perturbable but not segment-safe when the four edges opposite it are all segments. Whatever it moves in the facet, at least one segment becomes non-Delaunay.

Δ is said to be *removable* if there exits a vertex $p \in \Delta$, such that:

- p is perturbable; and
- there exists a perturbation of p which is segment-safe.

Otherwise, Δ is *unremovable*.

If Δ is unremovable. We introduce a new point v_b, called *break point*, it is chosen as follows:

- If the vertices of Δ are affinely independent. Let S_Δ be the common sphere shared by them. v_b is inside S_Δ.
- If the vertices of Δ are affinly dependent, i.e., four of them are coplanar. Let C_Δ be the common circle shared by the four vertices. v_b is coplanar with the four vertices and inside C_Δ.

v_b will break the local degeneracy (see Figure 6).

A break point may encroach upon one or more segments and subfaces of X_2. In this case, it will not be inserted. Instead, the following *boundary protection* procedure is called:

1. For each encroached segment, add its perturbed circumcenter to X_2. Updating $\mathcal{D}_2$, X_2 accordingly;

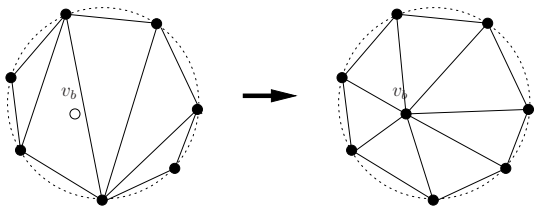

Fig. 6. The break point. On the left is a set of locally degenerate points and one of its Delaunay triangulations. v_b is a break point for removing the local degeneracy. On the right is the unique Delaunay triangulation after v_b is inserted.

2. For each encroached subface, compute its perturbed circumcenter x. If x encroaches upon any segments, it is not inserted, goto step 1 to split the encroached segments. Otherwise, add x to X_2. Updating $\mathcal{D}_2$, X_2 accordingly;
3 Call the Delaunay segment recovery algorithm to recover all missing segments of X_2 in $\mathcal{D}_2$.

The complete algorithm is listed below:

Algorithm *Local Degeneracy Removal*
Input: $\mathcal{D}_1$, X_1.
Output: $\mathcal{D}_2$, X_2.
initialize:
  $\mathcal{D}_2 := \mathcal{D}_1$, $X_2 := X_1$;
repeat:
  form a queue Q of local degeneracies of $\mathcal{D}_2$;
  while $Q \neq \emptyset$ **do**
    remove a local degeneracy Δ from Q;
    if Δ is removable **then**
      Remove Δ by a small perturbation;
    else
      Compute a v_b of Δ;
      if v_b encroaches upon any segment and subface **then**
        Push Δ into Q;
        Call the boundary protection procedure;
      else
        Insert v_b to break Δ;
        update $\mathcal{D}_2$, X_2;
      endif
    endif
  endwhile
until $\mathcal{D}_2$ contains no local degeneracy;

In our implementation, we choose v_b be the circumcenter of the common sphere of Δ. It is important that v_b should not create a new local degeneracy with other

existing vertices. This can be checked before inserting the point. If so, we should not insert v_b but choose another location. Notice that such case only can happen when the input PLC is highly symmetric, e.g., in Figure 5. A simple strategy is to add a randomized perturbation on each v_b. After the perturbation, the probability to create a symmetric point configuration again is nearly zero. Due to that reason, the points added for boundary protecting are perturbed, too.

Some break points may locate outside X_2. They will be removed after the CDT of X_2 is constructed.

The termination of the local degeneracy removal algorithm is due to the fact that the number of unremovable local degeneracies can only be decreased and no new local degeneracies are created by break points and segment protecting points.

7 Facet Recovery

The segment recovery and local degeneracy removal algorithms have produced X_2 such that it has a CDT (guaranteed by Theorem 2). Let $\mathcal{T}$ be a CDT of X_2. Generally, $\mathcal{D}_2 \neq \mathcal{T}$ because some subfaces of $\mathcal{T}$ are non-Delaunay faces and penetrated by edges of $\mathcal{D}_2$. This section describes an algorithm which incrementally transforms $\mathcal{D}_2$ into $\mathcal{T}$. No additional points are needed in the transformation.

The general steps of our algorithm are similar to [Shewchuk02]. At initialization, let $\mathcal{T}^{(0)} := \mathcal{D}_2$; add all missing subfaces into a queue Q. The algorithm starts to recover the subfaces in Q until Q is empty. At each step i, the algorithm recovers a set of missing subfaces in $\mathcal{T}^{(i)}$, update $\mathcal{T}^{(i)}$ into $\mathcal{T}^{(i+1)}$. After m steps Q is empty, and $\mathcal{T} = \mathcal{T}^{(m)}$.

At step $i (i < m)$, several missing subfaces are recovered together. We define a *missing region* Ω to be a set of coplanar subfaces of X_2 such that

- all subfaces of Ω belong to one facet of X_2;
- the boundary edges of Ω are edges of $\mathcal{T}^{(i)}$; and
- the internal edges of Ω are missing in $\mathcal{T}^{(i)}$.

Hence Ω is a connected set of missing coplanar subfaces. It may not simply connected, i.e., Ω can contain a hole inside (see Figure 7 (a)). Each missing subface belongs to one missing region. A facet can have more than one missing regions.

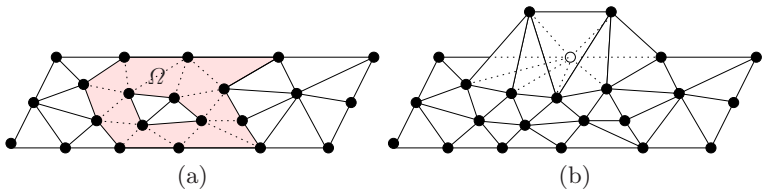

Fig. 7. (a) The shaded area highlights a non-simply connected missing region Ω. (b) A cavity C at (step i) is illustrated.

When a Ω is found, the *formcavity* procedure (described below) forms two cavities at each side of Ω. Each cavity is bounded by subfaces of Ω and faces of $\mathcal{T}^{(i)}$ (see Figure 7 (b)).

1. Find all tetrahedra in $\mathcal{T}^{(i)}$ that intersect the relative interior of Ω, delete them from $\mathcal{T}^{(i)}$. This creates a hole inside $\mathcal{T}^{(i)}$.
2. Insert the missing subfaces of Ω into the hole to split it into two separated cavities, one at each side of Ω.

Each cavity C is filled with a set of new tetrahedra by the following *cavity retetrahedralization* procedure. Let V be the set of vertices of C.

1. Verify C, expand C if it is necessary.
   (1) Form a queue Q containing all non-strongly Delaunay faces of C in V;
   (2) For each face $\sigma \in Q$ and σ still in C,
   let t be the tetrahedron adjacent to C and holds σ;
   remove σ from C;
   for each face Δt of t, $\Delta t \neq \sigma$ **do**
   if Δt is a face of C **then** remove Δt from C;
   else add Δt into C; **end**
   end
   Update C, V;
   (3) Repeat (1) if some faces of C are not strongly Delaunay in V.
   See Figure 8 for an example of the expansion of C.
2. Retetrahedralize C.
   (1) Construct a Delaunay tetrahedralization $\mathcal{D}_C$ of the vertices of the C.
   (2) Identify faces of C in $\mathcal{D}_C$, mark each tetrahedron of $\mathcal{D}_C$ to be "inside" or "outside".
   (3) Remove tetrahedra marked as "outside" from $\mathcal{D}_C$, and fill the remaining tetrahedra into C.
   Figure 9 illustrates a two dimensional example of the retetrahedralization procedure.

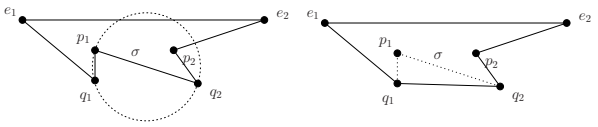

Fig. 8. The expansion of the cavity (illustrated in 2D). Left: e_1e_2 is the segment going to recover. Edges below e_1e_2 is the faces of the cavity C. p_1q_2 is not strongly Delaunay. Right: C is expanded by removing two edges p_1q_1 and p_1q_2 from C and adding one edge q_1q_2 into C.

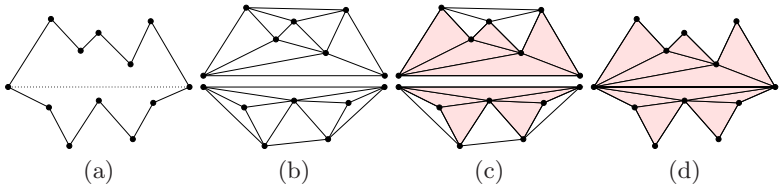

Fig. 9. Cavity triangulation (illustrated in 2D). (a) Two initial cavities separated by a constraining segment. (b) The two Delaunay triangulations constructed at each side of the segment. (c) Mark triangles as "inside" or "outside". (d) Remove "outside" triangles.

The complete facet recovery algorithm is summaried as follows:

Algorithm *Facet Recovery*
Input: $\mathcal{D}_2$, X_2.
Output: The CDT $\mathcal{T}$ of X_2.
initialize:
  $\mathcal{T} := \mathcal{D}_2$;
Repeat:
  form a queue Q of missing subfaces in $\mathcal{T}$;
  while $Q \neq \emptyset$ **do**
    remove an unrecovered subface f from Q;
    form a missing region Ω containing f;
    form two cavities C_1, C_2 by formcavity subroutine;
    for each $C_i, i = \{1, 2\}$ **do**
      Call cavity retetrahedralization subroutine;
    end
  end
until no subfaces are missing in $\mathcal{T}$.

Theorem 4. *The facet recovery algorithm terminates.*

Proof: We show that the cavity retetrahedralization subroutine will always succeed, hence the missing region found at each step can be recovered without getting stuck.

The step 1 (face verification) of the cavity retetrahedralization algorithm guarantees the step 2 can be correctly executed since every strongly Delaunay simplex of V will appear in any Delaunay tetrahedralization of V. Hence the face identification in step 2 must be successful. What remains is to prove two issues in the face verification step: (1) the expansion of C terminates; and (2) the missing subfaces due to the expansion of C can be recovered later.

Let Ψ be the set of faces of $\mathcal{D}_2$ such that no face of Ψ is crossed by any subfaces of X_2. Clearly, $\Psi \neq \emptyset$ and any face $\Delta t \in \Psi$ is strongly Delaunay and exists in any $\mathcal{T}^{(i)}$. The set Ψ limits the expansion of C, i.e., C stops expanding at σ when $\sigma \in \Psi$.

Let $Vol(C)$ be the inside volume of C. During the expansion of C some subfaces are missing. Notice that the missing region Ω' formed by these missing subfaces is completely inside C, hence $Vol(C') < Vol(C)$, where C' is the cavity formed from Ω'. This relation holds if the expansion of C' still causes some other subfaces missing and results another new cavity. Such sequence will terminate since the value of

$Vol()$ can not be negative. ■

Theorem 5. *$\mathcal{T}$ created by the facet recovery algorithm is a CDT.*

Proof: We first show $\mathcal{T}^{(1)}$ is a CDT. $\mathcal{T}^{(1)}$ is the result of cavity retetrahedralization algorithm on $\mathcal{T}^{(0)}$. Tetrahedra of $\mathcal{T}^{(1)}$ which are outside and not adjacent to the cavities remain Delaunay. Let t be a tetrahedron created inside a cavity C, S_t is its circumsphere. Let t' be another tetrahedron in $\mathcal{T}^{(1)}$ sharing a face σ with t, v be the vertex of t' opposite to σ. We have the following cases:

(1) σ is a subface. Then t is constrained Delaunay even if S_t encloses v, i.e, the inside of t is not visible by v, otherwise at least a segment is non-Delaunay and can not exist in $\mathcal{T}^{(0)}$;
(2) σ is a face inside C, then S_t must not enclose v (guaranteed by the cavity retetrahedralization algorithm).
(3) σ is a face on C, then σ is not a subface, S_t must not enclose v. Otherwise, $\mathcal{T}^{(0)}$ is not a Delaunay tetrahedralization since the circumsphere of t' is not empty, i.e., it encloses the point of t opposite to σ.

By induction, after step i, $i > 1$, $\mathcal{T}^{(i)}$ is a CDT. Now we can show $\mathcal{T}^{(i+1)}$ is a CDT by using the similar arguments as above. The only difference is in the case (3) which is stated below:

(3) σ is a face on C, then we have two cases,
   a. σ is not a subface, S_t must not enclose v. Otherwise, $\mathcal{T}^{(i)}$ is not a CDT since the circumsphere of t' encloses the point of t opposite to σ which is also visible from the inside of t'.
   b. σ is a subface, then t is constrained Delaunay even if S_t encloses v.

Thus on the finish of the facet recovery algorithm $\mathcal{T} = \mathcal{T}^{(m)}$ is a CDT. ■

8 Experimental Results

This algorithm has been implemented and in our 3D quality Delaunay mesh generator - TetGen [Si04b] (`http://tetgen.berlios.de`). We have tested the program with a number of examples not only having simple but also relatively complicated geometries. The algorithm runs rather efficiently compares to the old version of TetGen which uses gift-wrapping algorithm. For example, for a cavity having around 300 faces and 150 vertices, the gift-wrapping algorithm needs few minites to finish while the cavity retetrahedralization algorithm finishes in less than 1 second. To test the stability of the algorithm as well as our implementation, we purposely chose some models which the surface meshes are rather badly discretized. They are most likely to pull down the program if the algorithm has defect. On most of these models the program successfully produced CDTs as long as the surface meshes are PLCs.

The following two PLC models can be freely downloaded from Inria's large repository `http://www-rocq1.inria.fr/gamma`.

The Cup-holder (in Figure 10) has a simple geometry (918 vertices, 1848 triangles). 1261 points are added (262 break points and 999 protecting points). There are

157 missing subfaces, the biggest size of a cavity contains 40 faces. Moreover, the effect of removing local degeneracies can be visualized on the right picture.

The monster4 (in Figure 11) consists of 1392 vertices, 2784 triangles. It has complicated distribution of segments and facets as shown in the left picture. Our algorithm added 2782 Steiner points (502 break points and 2226 protecting points).

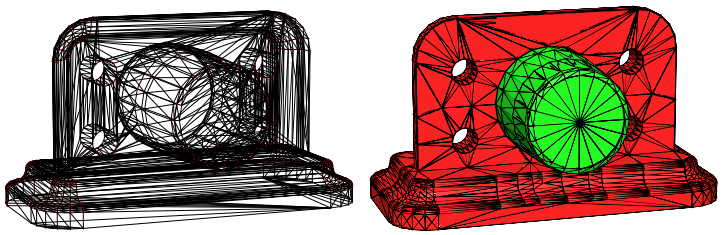

Fig. 10. Example 1 (Cup-holder). Left: the PLC. Right: the CDT.

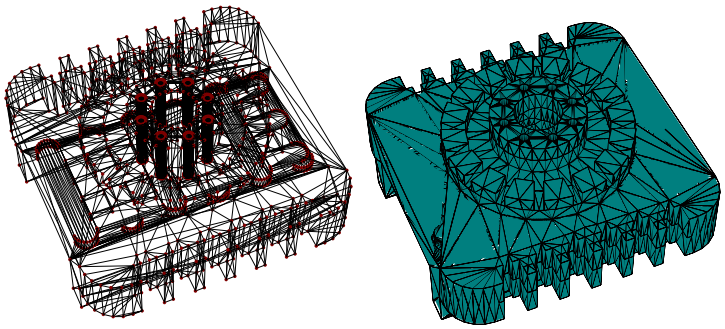

Fig. 11. Example 2 (monster4). Left: the PLC. Right: the CDT.

The geometry of Figure 12 is a human heart inside a body. The surface mesh of the heart is internal boundary separating two regions. Hence this model is made up of multiple domains. In spite of the complexities in the geometries, the point set contain no local degeneracy. Hence the CDT only require few additional points (compare to the input number of points) for protecting segments. The CDT is shown in the middle. One can see that the tetrahedra of the CDT are usually very long and skinny. The picture on the right shows the better quality tetrahedra obtained by performing a Delaunay refinement on the CDT. The number of additional points for improving the mesh quality are much bigger than that of getting a CDT.

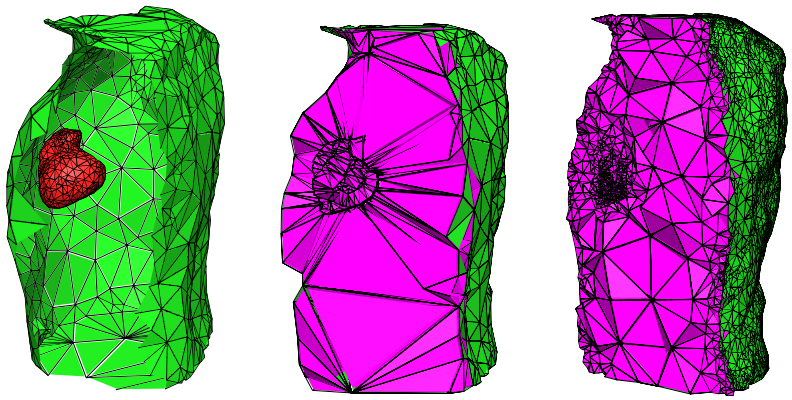

Fig. 12. Example 3 (Heart). Left: the PLC. Middle: the CDT. Right: the quality mesh. A vertical cut has been made on the meshes so that the internal boundaries can be seen.

Acknowledgements

This work is supported by the pdelib project of Weierstrass Institute for Applied Analysis and Stochastics. We are grateful to the reviewers for the valuable suggestions and comments, especially for the pointing out a potential problem in a proof of the first draft.

[Schoenhardt28] Schönhardt E (1928) Über die Zerlegung von Dreieckspolyedern in Tetraeder. Mathematische Annalen 98:309–312

[Bagemihl48] Bagemihl F (1948) On Indecomposable Polyhedra. American Mathematical Monthly 55: 411–413

[Chazelle84] Chazelle B. (1984) Convex Partition of Polyhedra: A lower Bound and Worst-case Optimal Algorithm. SIAM Journal on Computing 13(3): 488–507

[Lee86] Lee D T, Lin A K (1986) Generalized Delaunay Triangulations for Planar Graphs. Discrete Comput Geom 1:201–217

[Chew89] Chew P L(1989) Constrained Delaunay Triangulation. Algorithmica 4(1): 97–108

[Edelsbrunner90] Edelsbrunner H, Mücke M P (1990) Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithm. ACM Transactions on Graphics 9(1): 66–104

[Edelsbrunner96] Edelsbrunner H, Shah N R (1996) Incremental Topological Flipping Works for Regular Triangulations. Algorithmica 15: 223–241

[Ruppert92] Ruppert J (1992) On the Difficulty of Triangulating Three-dimensional Non-convex Polyhedra. Discrete Comput Geom 7: 227–253

[Ruppert95] Ruppert J (1995) A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. J Algorithms 18(3): 548–585

[Weatherill94] Weatherill N P, Hassan O (1994) Efficient Three-Dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints. Int. J. Numer. Meth. Engng 37: 2005–2039

[Miller96] Miller G L, Talmor D, Teng S H, Walkington N, Wang H (1996) Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening. In: Proc. of 5th Intl. Meshing Roundtable

[Shewchuk98a] Shewchuk J R (1998) A Condition Guaranteeing the Existence of Higher-Dimensional Constrained Delaunay Triangulations. In: Proc. of the 14th Annu. Sympos. Comput. Geom., 76–85, Minneapolis, Minnesota

[Shewchuk98b] Shewchuk J R (1998) Tetrahedral Mesh Generation by Delaunay Refinement. In: Proc. of the 14th Annu. Sympos. Comput. Geom.

[Shewchuk00] Shewchuk J R (2002) Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations. In: Proc. of the 16th Annu. Sympos. Comput. Geom.

[Shewchuk02] Shewchuk J R (2002) Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery. In: Proc. of 11th Intl. Meshing Roundtable

[Shewchuk03] Shewchuk J R (2003) Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips. In: Proc. 19th Annu. Sympos. Comput. Geom.

[Pebay98] Pébay P (1998) A Priori Delaunay-Conformity. In: Proc. of 7th Intl Meshing Roundtable, SANDIA

[Murphy00] Murphy M, Mount D M, Gable C W (2000) A Point-Placement Strategy for Conforming Delaunay Tetrahedralization. In: Proc. of the 11th Annu. Sympos. on Discrete Algorithms

[Cohen-Steiner02] Cohen-Steiner D, Colin de Verdière E, Yvinec M (2002) Conforming Delaunay Triangulations in 3D. In: Proc. of 18th Annu. Sympos. Comput. Geom. Barcelona

[George03] George P L, Borouchaki H, Saltel E (2003) 'Ultimate' Robustness in Meshing an Arbitrary Polyhedron. Int. J. Numer. Meth. Engng 58: 1061–1089

[Rambau03] Rambau J. (2003) On a Generalization of Schönhardt's Polyhedron. MSRI Preprint 2003-13

[Du04] Du Q, Wang D (2004) Constrained Boundary Recovery for Three Dimensional Delaunay Triangulations. Int. J. Numer. Meth. Engng 61: 1471–1500

[Cheng04] Cheng S W, Dey T K, Ramos E A, Ray T (2004) Quality Meshing for Ployhedra with Small Angels. In: Proc. 20th Annu. ACM Sympos. Comput. Geom.

[Pav04] Pav S, Walkington N (2004) A Robust 3D Delaunay Refinement Algorithm. In: Proc. Intl. Meshing Roundtable.

[Si04a] Si H, Gärtner K (2004) An Algorithm for Three-Dimensional Constrained Delaunay Triangulations. In: Proc of 4th Intl. Conf. on Engineering Computational Technology, Lisbon

[Si04b] Si H (2004) TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, v1.3 User's Manual. WIAS Technical Report No. 9