# Vertex Deletion for 3D Delaunay Triangulations

Kevin Buchin[1], Olivier Devillers[2], Wolfgang Mulzer[3],
Okke Schrijvers[4], and Jonathan Shewchuk[5]

[1] Technical University Eindhoven, The Netherlands www.win.tue.nl/∼kbuchin
[2] INRIA Sophia Antipolis – Méditerranée, France inria.fr/sophia/members/Olivier.Devillers
[3] Freie Universität Berlin, Germany page.mi.fu-berlin.de/mulzer
[4] Stanford University, USA www.okke.info
[5] University of California at Berkeley, USA www.cs.berkeley.edu/∼jrs

**Abstract.** We show how to delete a vertex $q$ from a three-dimensional Delaunay triangulation $\mathrm{DT}(S)$ in expected $O(C^{\otimes}(P))$ time, where $P$ is the set of vertices neighboring $q$ in $\mathrm{DT}(S)$ and $C^{\otimes}(P)$ is an upper bound on the expected number of tetrahedra whose circumspheres enclose $q$ that are created during the randomized incremental construction of $\mathrm{DT}(P)$. Experiments show that our approach is significantly faster than existing implementations if $q$ has high degree, and competitive if $q$ has low degree.

## 1 Introduction

Some geometric applications require the ability to delete a vertex from a Delaunay triangulation. An early algorithm by Chew [9] for generating guaranteed-quality triangular meshes uses Delaunay vertex deletion to obtain a better bound on the minimum angle than is achieved by similar algorithms that do not use vertex deletion, and the same principle has been exploited by mesh generators that generate better-quality tetrahedra by occasionally deleting vertices from a three-dimensional Delaunay triangulation [7, Section 14.5]. Another application of Delaunay vertex deletion is interactive data cleaning, in which a user desires to remove outlier vertices from a triangulation used to interpolate data.

Let $S$ be a finite set of points in $\mathbb{R}^2$ or $\mathbb{R}^3$, and let $\mathrm{DT}(S)$ denote its Delaunay triangulation. We study how to *delete* a vertex from $\mathrm{DT}(S)$ while maintaining the Delaunay property of the triangulation. That is, given a point $q \in S$, we wish to transform $\mathrm{DT}(S)$ into $\mathrm{DT}(S \setminus \{q\})$ quickly.

In two dimensions, Delaunay deletion is well understood. By 1990, several algorithms were known that delete a vertex $q$ with degree $d$ in optimal $O(d)$ time [1, 8], and fast, practical implementations are available now [13]. In three dimensions, there is still room for improvement. In theory, the best methods known are *triangulate and sew* and *ear queue*. The ear-queue algorithm has worst-case running time $O(k \log d)$ where $k$ is the number of new tetrahedra created. The triangulate-and-sew algorithm triangulates the set $P$ of vertices neighboring $q$ in $\mathrm{DT}(S)$ with the standard randomized incremental construction (RIC) algorithm, then takes the tetrahedra adjoining $q$ in $\mathrm{DT}(P)$ and sews them into the cavity. This method runs in expected $O(d \log d + C(P))$ time, where

$d = |P|$ and $C(P)$ is the expected number of tetrahedra created during the randomized incremental construction of $\mathrm{DT}(P)$. These two complexities, $O(k \log d)$ and $O(d \log d + C(P))$, are not comparable as $\Omega(d) \leq k \leq C(P) \leq O(d^2)$.

The main innovations behind our algorithms are fast methods for point location, so all the point location steps of the incremental construction take expected total time linear in $d$, and a vertex insertion procedure whose cost is $C^{\otimes}(P)$, the expected number of tetrahedra *in conflict with* $q$ created during the randomized incremental construction of $\mathrm{DT}(P)$. Our complexity $O(d+C^{\otimes}(P)) = O(C^{\otimes}(P))$ is always better than $O(d \log d+C(P))$ and better than the ear queue algorithm in the usual circumstance that $C^{\otimes}(P) = o(k \log d)$. Moreover, it uses less complicated numerical predicates than the ear queue, and therefore it is easier to implement robustly. A prototype implementation of our algorithm compares favorably with existing codes, particularly if $q$ has high degree.

## 2 Related work

*Existing Algorithms.* The literature contains several algorithms for Delaunay vertex deletion. All of them begin by deleting $q$ and the simplices adjoining $q$, thereby evacuating a star-shaped *cavity* in the DT, which must be retriangulated. Some vertex deletion algorithms are primarily concerned with the asymptotic running time as a function of the degree $d$ of $q$, but the average vertex degree in a two-dimensional DT is less than six, so some authors emphasize the speed when $d$ is small. The cavity's Delaunay triangulation always has size $\Theta(d)$ in 2D, but in 3D its size may be as small as $\Theta(d)$ or as large as $\Theta(d^2)$.

The *gift-wrapping* or *boundary completion* algorithm constructs one triangle or tetrahedron at a time by choosing a known facet (edge or triangle) $f$ and finding a vertex $p \in P$ so that $f$ and $p$ together form a Delaunay triangle or tetrahedron. Its worst-case running time is $\Theta(d^2)$ in 2D [13] and $\Theta(d^3)$ in 3D.

The *ear queue algorithm* [12] is a gift-wrapping algorithm that uses a priority queue to quickly identify an ear that can be cut off the star-shaped cavity. Each ear is assigned a priority proportional to a numerical quantity called the *power* of the circumsphere with respect to the deleted vertex $q$; the highest-priority ear is guaranteed to be Delaunay. The algorithm runs in $O(d \log d)$ time in 2D and $O(k \log d)$ time in 3D, where $k$ is the number of new tetrahedra created. Unfortunately, it requires a new geometric predicate that compares the powers of two ears. This makes the code less generic and more difficult to make robust and efficient.

The *flip algorithm* connects all the facets on the cavity boundary to a single vertex in $P$, then performs a sequence of *flips* that replace simplices with other simplices. In 2D, the flip algorithm finds the Delaunay triangulation in $O(d^2)$ time [16], but in 3D, the flip algorithm does not always work; it can get stuck in a non-Delaunay triangulation from which no flip can make further progress [15].

In 2D, two algorithms are known that run in linear time, which is optimal. Aggarwal, Guibas, Saxe, and Shor [1] describe an algorithm that runs in deterministic $O(d)$ time, but it is complicated and not practical. In a classic paper that

introduced the randomized algorithm analysis technique now known as *backward analysis*, Chew [8] proposes a simple, practical, randomized algorithm that runs in expected $O(d)$ time. The algorithm, which we call *backward reinsertion*, combines RIC with a backward point location method. The algorithm of Aggarwal et al. does not appear to generalize to 3D, but in this paper we generalize backward reinsertion to 3D.

Devillers [13] has explicitly constructed optimal algebraic decision trees for deleting vertices of degree at most 7 from 2D Delaunay triangulations. This approach, called *low-degree optimization*, obtains notable speedups for vertices of small degree. We do not foresee it being extended to 3D, where the complexity of the decision trees grows very quickly.

The *triangulate-and-sew algorithm* retriangulates the cavity by computing $\mathrm{DT}(P)$ from scratch, taking the subset of simplices in $\mathrm{DT}(P)$ that lie inside the cavity, and sewing them into the cavity to obtain $\mathrm{DT}(S \setminus \{q\})$. If we use a RIC algorithm to compute $\mathrm{DT}(P)$, triangulate-and-sew runs in $O(d \log d + C(P))$ time, whether in 2D or 3D. This approach may create many simplices that are unnecessary because they lie outside the cavity or are not helpful in computing the final simplices inside the cavity. We address this drawback below.

*Existing Implementations.* For Delaunay vertex deletions in 2D, CGAL [5] implements low-degree optimization for vertices of degree 7 or less. For higher degrees, it uses flipping. In 3D, the current implementation offers triangulate-and-sew. A previous version used a simplified ear queue algorithm with running time $O(dk)$ [12].

## 3 Preliminaries and Notation

We are given a finite point set $S \subseteq \mathbb{R}^3$ and its Delaunay triangulation $\mathrm{DT}(S)$, and we wish to delete the vertex $q \in S$ from $\mathrm{DT}(S)$, yielding $\mathrm{DT}(S \setminus \{q\})$. A point $p \in S$ is a *neighbor* of $q$ if $\mathrm{DT}(S)$ contains the edge $pq$. Let $P$ be the set of neighbors of $q$ in $\mathrm{DT}(S)$. Let $d = \mathrm{d}^\circ(q, \mathrm{DT}(S)) = |P|$ be the *degree* of $q$ in $\mathrm{DT}(S)$.

We use a *randomized incremental construction* (RIC) algorithm to compute $\mathrm{DT}(P)$. The standard RIC algorithm successively inserts the points in $P$ into a Delaunay triangulation, one by one, in an order determined by a random permutation $p_1, p_2, \ldots, p_d$ of $P$. For $i = 1, \ldots, d$, let $P_i = \{p_1, \ldots, p_i\}$ contain the first $i$ points of the permutation. The standard RIC constructs $\mathrm{DT}(P)$ by successively inserting each $p_i$ into $\mathrm{DT}(P_{i-1})$. A tetrahedron in $\mathrm{DT}(P_{i-1})$ is said to *conflict* with $p_i$ if its circumsphere encloses $p_i$. The algorithm identifies all the *conflict tetrahedra* in three steps. First, a method called *point location* identifies one tetrahedron $\Delta$ that conflicts with $p_i$. Second, the algorithm finds all the other tetrahedra in $\mathrm{DT}(P_{i-1})$ that conflict with $p_i$ by a depth-first search from $\Delta$. This search treats $\mathrm{DT}(P_{i-1})$ as a graph in which each tetrahedron acts as a graph node and two nodes are connected by a graph edge if the corresponding tetrahedra share a triangular face. Third, the conflict tetrahedra are all deleted.

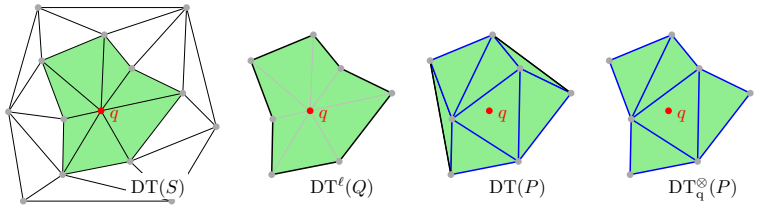



**Fig. 1.** A DT of $S$, a link DT for $q$, a DT of $q$'s neighbors, and $q$'s conflict DT.

The union of the conflict tetrahedra is a *cavity* which we retriangulate with tetrahedra adjoining $p_i$. This step is called *structural change*. The expected cost of the structural change, denoted $C(P)$, is obtained by summing the cost of inserting a random point into $\mathrm{DT}(P_i)$ for $i = 1, \ldots, d$.

Point location is usually the most difficult part of incremental construction algorithms; we will discuss it at length later. We will use the special nature of Delaunay vertex deletion both to speed up point location and to reduce the number of structural changes we must make. We will also use a variant of RICs that inserts points in batches.

Let $Q = P \cup \{q\}$ and $Q_i = P_i \cup \{q\}$ for $i = 1, \ldots, d$. The *link DT*, denoted $\mathrm{DT}^\ell(Q_i)$, is the subset of $\mathrm{DT}(Q_i)$ containing only the tetrahedra adjoining $q$ and their faces, as illustrated in Figure 1. The name stems from the fact that the boundary faces of $\mathrm{DT}^\ell(Q_i)$ form a triangulation of a topological sphere. The *conflict DT*, denoted $\mathrm{DT}^\otimes_q(P_i)$, is the subset of $\mathrm{DT}(P_i)$ containing only the tetrahedra whose circumspheres enclose $q$. Observe that the boundaries of $\mathrm{DT}^\otimes_q(P_i)$ and $\mathrm{DT}^\ell(Q_i)$ are identical. The expected cost of the structural change restricted to the tetrahedra of $\mathrm{DT}^\otimes_q$ is denoted $C^\otimes(P)$.

## 4 Algorithm

Our algorithm uses randomized incremental construction to compute $\mathrm{DT}^\otimes_q(P)$, the tetrahedra that conflict with $q$ in the DT of $q$'s neighbors, and uses it to fill the cavity evacuated by $q$'s deletion. To insert each new point $p_i$ into $\mathrm{DT}^\otimes_q(P_{i-1})$, we need to quickly identify a tetrahedron that conflicts with $p_i$. For this, we maintain the *link DT* $\mathrm{DT}^\ell(Q_i)$: the DT of the points $P_i \cup \{q\}$, restricted to the tetrahedra adjoining $q$. The boundaries of $\mathrm{DT}^\ell(Q_i)$ and $\mathrm{DT}^\otimes_q(P_i)$ are identical, so we can use any edge of $\mathrm{DT}^\ell(Q_i)$ adjoining $p_i$ to find a conflict tetrahedron in $\mathrm{DT}^\otimes_q(P_{i-1})$.

To obtain the sequence $\mathrm{DT}^\ell(Q_4), \ldots, \mathrm{DT}^\ell(Q_d)$, we use the *reverse deletion trick* [8]. By construction, $\mathrm{DT}^\ell(Q) \subseteq \mathrm{DT}(S)$. We remove the points $p_d, \ldots, p_5$ in that order from $\mathrm{DT}^\ell(Q)$. If the deletion order is sufficiently random, this process can be implemented efficiently, as the boundary of $\mathrm{DT}^\ell(Q)$ behaves like a 2D Delaunay triangulation. Each time we remove a point $p_i$ from $\mathrm{DT}^\ell(Q_i)$, we store a *guide* for $p_i$, denoted $\mathrm{guide}(p_i)$, to help the point location of $p_i$ in $\mathrm{DT}^\otimes_q(P_{i-1})$; see Figure 2. The guide is usually a neighbor of $p_i$ in $\mathrm{DT}^\ell(Q_i)$, but different variants of the algorithm use different guides.

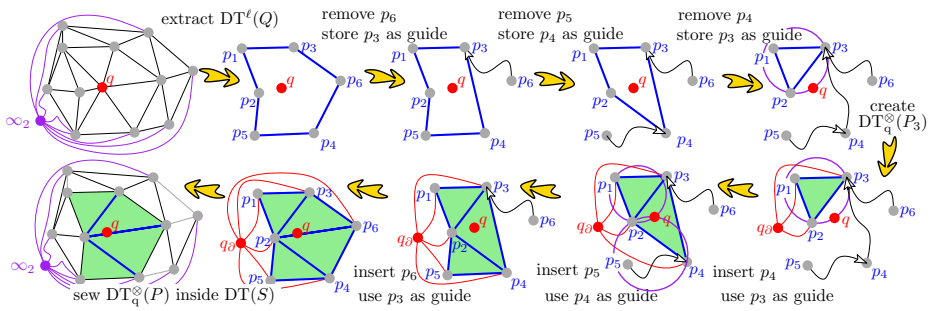



**Fig. 2.** An illustration of the deletion and reinsertion process in 2D.

We now describe how to use guide($p_i$) when inserting $p_i$ into $\mathrm{DT}^\otimes_q(P_{i-1})$. The point location uses two steps: (i) finding the tetrahedron adjoining guide($p_i$) that intersects the line segment guide($p_i$)$p_i$; and (ii) walking to the tetrahedron that contains $p_i$. The second step visits only tetrahedra that are destroyed during insertion, so its cost can be charged to the structural change $C^\otimes(P)$. The time for the first step is proportional to $\mathrm{d}^\circ(\text{guide}(p_i), \mathrm{DT}^\ell(Q_{i-1}))$. This depends on the exact nature of the insertion order, and we will discuss it below.

**The Link Delaunay Triangulation.** All the tetrahedra of $\mathrm{DT}^\ell(Q_i)$ share the vertex $q$, so the *link* of $q$ (i.e., the triangles in $\mathrm{DT}^\ell(Q_i)$ that do not contain $q$) has the topology of a 2D sphere. Hence, we can represent $\mathrm{DT}^\ell(Q_i)$ as a 2D triangulation. The triangulation $\mathrm{DT}^\ell(Q_d)$ can be extracted from $\mathrm{DT}(S)$ in $O(d)$ time by a simple traversal of the tetrahedra adjoining $q$. To maintain $\mathrm{DT}^\ell(Q_i)$ under deletion, we can use any ordinary 2D Delaunay algorithm while replacing the in-circle test w.r.t. a triangle by the in-sphere test w.r.t. the tetrahedron formed by the triangle and $q$; see Section 4.1. Correctness follows because we are looking for the triangles $t$ where the sphere passing through the vertices of $t$ and $q$ is empty.

**The Conflict Delaunay Triangulation.** Recall that we defined $\mathrm{DT}^\otimes_q(P_i)$ as the set of all tetrahedra in $\mathrm{DT}(P_i)$ that have $q$ in their circumsphere. Our goal is to prevent $\mathrm{DT}(P_i) \setminus \mathrm{DT}^\otimes_q(P_i)$ from being constructed. If we were to construct all of $\mathrm{DT}(P)$, we might create tetrahedra that would be discarded when we sew the cavity back into the original triangulation. In 3D, the number of unnecessary tetrahedra can be quadratic.

**Lemma 1.** *For any $d \in \mathbb{N}$, there is a $d$-point set $P \subseteq \mathbb{R}^3$ with $C(P) = \Omega(d^2)$ and $C^\otimes(P) = O(d)$.*

*Proof.* We take $P$ to be $d$ points on the three-dimensional *moment curve* $t \mapsto (t, t^2, t^3)$. It is well known that $\mathrm{DT}(P)$ has complexity $\Omega(d^2)$ and that all points of $P$ are on the lower part of the convex hull. Thus, if we take $q$ sufficiently far

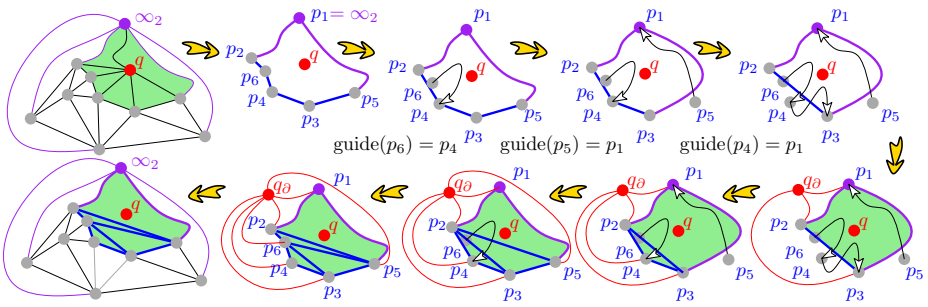



**Fig. 3.** 2D example of the deletion and reinsertion process with $q$ on the convex hull.

below $P$, then $q$ connects to all points of $P$ in $\mathrm{DT}(P \cup \{q\})$. When deleting $q$, $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P)$ contains only $O(d)$ infinite tetrahedra, but the full triangulation $\mathrm{DT}(P)$ consists of $\Omega(d^2)$ tetrahedra. □

### 4.1 Managing the Boundaries

During the randomized incremental construction of a triangulation, we need to take care of handling the boundary and the adjacencies between boundary facets. In a full Delaunay triangulation, this boundary is the convex hull, while in an intermediate triangulation such as $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P)$ it may be not convex.

To avoid complicated code for all the special cases, a classic approach adds a *dummy vertex* $\infty$ and creates for each facet $f$ of the convex hull a tetrahedron between $f$ and $\infty$ [3]. Thus, adjacencies between convex hull facets are managed as adjacencies between infinite tetrahedra. The circumsphere of an infinite tetrahedron is defined as the half space that is delimited by the plane through its finite facet, the side of the plane is determined by the tetrahedron orientation. The construction algorithm needs no special code for infinite tetrahedra, except inside the in-sphere predicates.

In our setting, we have three different triangulations: $\mathrm{DT}(S)$, $\mathrm{DT}^{\ell}(Q_i)$, and $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$. The boundary of $\mathrm{DT}(S)$ is managed as above, using a dummy vertex $\infty_3$ (the index 3 emphasizes the dimension). The triangulation $\mathrm{DT}^{\ell}(Q_i)$ does not have a boundary, since it is just the link of $q$ in some triangulation. However, note that if $q$ lies on the convex hull of $S$, then $\infty_3$ is a vertex of $\mathrm{DT}^{\ell}(Q)$. Finally, $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$ is a 3D triangulation with boundary $\mathrm{DT}^{\ell}(Q_i)$; to handle this boundary, we introduce another dummy vertex $q_\partial$ (pronounced "$q$ boundary") that forms a tetrahedron with each face of the boundary of $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$.

With this approach, we can use a standard deletion algorithm for each $\mathrm{DT}^{\ell}(Q_i)$ and a standard construction algorithm for each $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$. All special treatment goes into the in-sphere and in-circle predicates, see below. Figure 2 shows the deletion and reinsertion process when $q$ is not on the convex hull of $S$. In Figure 3, the point $q$ is on the convex hull of $S$, and $\infty_2$ and $q_\partial$ must interact.

**In-circle predicate for the link DT.** Let incircle$(a, b, c, w)$ be the predicate that is true if and only if either $w$ is inside the circumcircle for the triangle $abc$ and $abc$ is positively oriented or $w$ is outside the circumcircle and $abc$ is negatively oriented. The predicate insphere$(a, b, c, d, w)$ is defined analogously for the point $w$ and the tetrahedron $abcd$.

A triangle $abc$ belongs to the link DT and is positively oriented if and only if the sphere through $qabc$ is empty and the tetrahedron $qabc$ is positively oriented. More precisely, the incircle test incircle$^{\ell}(a, b, c, w)$ for the 2D deletion algorithm is implemented as insphere$(q, a, b, c, w)$. If one of $a$, $b$, $c$ or $w$ is $\infty_3$, the usual way of solving it is used, that is, insphere$(q, a, b, \infty_3, w)$ is true if tetrahedron $qabw$ is positively oriented.

Although $q$ initially lies inside of $\mathrm{DT}(Q_i)$, it might end up on the boundary during the deletion process (e.g., in Figure 2 at the deletion of $p_4$). We defined insphere in such a way that this does not cause a problem: if a tetrahedron has negative orientation, we want the point to be outside of its circumsphere (in Figure 2, when deleting $p_4$, the outside of the circle through $p_2p_3q$ does not contain $p_1$, and $p_2p_3$ is a boundary face of $\mathrm{DT}^{\ell}(Q_3)$).

**In-sphere predicate for the conflict DT.** Let $abcd$ be a positively oriented tetrahedron of $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$. By definition, insphere$(a, b, c, d, q)$ holds. If $q_\partial \notin \{a, b, c, d\}$, the predicate insphere$^{\otimes}(a, b, c, d, w)$, used to compute $\mathrm{DT}^{\otimes}_{\mathrm{q}}$, is defined as insphere$(a, b, c, d, w)$ (notice that $a$, $b$, $c$, or $d$ might be $\infty_3$). If, without loss of generality, $q_\partial = d$, we consider $bacd'$, the neighboring tetrahdron of $abcq_\partial$ in $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$. Let $C$ be the circumsphere of $bacd'$. Then $q$ lies inside of $C$. Consider moving $C$ in the pencil of spheres through $abc$ in the direction that places $d'$ outside. Since $abc$ is a face of the boundary of $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$, the moving sphere will encounter $q$ before any other point, so the sphere through $bacq$ is empty, and we consider it as the circumsphere of $abcq_\partial$. Again, if $q$ is not on the same side of $abc$ as $d'$, the conflict zone is actually the outside of the ball. More formally, insphere$^{\otimes}(a, b, c, q_\partial, w)$ is defined as insphere$^{\otimes}(b, a, c, q, w)$. For an example in 2D refer to Figure 2: when $p_5$ is inserted, it is in conflict with $p_4p_2q_\partial$ and not with $p_2p_1q_\partial$ creating a non-convex angle on the boundary of $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_5)$. When $p_4$ is inserted, since $p_3p_2q$ is counterclockwise the disk circumscribing $p_3p_2q_\partial$ is the outside of the circumscribing circle of $p_2p_3q$, and $p_4$ is in conflict.

### 4.2 Main Algorithm

We now present several variations of our randomized incremental algorithm. All variants use the same framework, given in Algorithm DeleteVertex, but they differ in the implementation of ConstructDT in line 3. Some of our schemes achieve good worst-case performance, while others yield more practical implementations.

DeleteVertex($\mathrm{DT}(S), q$)

▷ Preprocessing
1 $\mathrm{DT}^\ell(Q) \leftarrow$ ConstructLinkDT($\mathrm{DT}(S), q$);
2 $P \leftarrow$ the neighbors of $q$;
▷ The actual implementation for filling the cavity
3 $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P) \leftarrow$ ConstructDT($\mathrm{DT}^\ell(Q), P, q$);
▷ Postprocessing
4 Sew the tetrahedra from $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P)$ into $\mathrm{DT}(S)$
using the correspondence between $\mathrm{DT}^\ell(Q)$ and the boundary of $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P)$;
5 Delete the tetrahedra of $\mathrm{DT}^\ell(Q)$;

It is plain to observe that the pre- and postprocessing takes time $O(|P|)$. Thus, the complexity lies in the recursive function ConstructDT.

We give two approaches for ConstructDT: *incremental backward reinsertion* (IBR) and a *biased randomized insertion order* (BRIO). IBR samples random points from $P$ one-by-one and updates $\mathrm{DT}^\ell(Q_i)$ before sampling a new point. BRIO uses a *gradation* of $P$, i.e., it batches the sampling process into *rounds*, and it updates $\mathrm{DT}^\ell(Q_i)$ only once all points of a round have been determined.

**Incremental Backward Reinsertion** The IBR-approach is given as Algorithm IBR-Construct. It samples a point $p_i$ from $\mathrm{DT}^\ell(Q_i)$, recursively constructs $\mathrm{DT}(P_{i-1})$, and then inserts $p_i$ into $\mathrm{DT}(P_{i-1})$ using guide($p_i$). The correctness of Algorithm IBR-Construct follows immediately by induction, but there are several choices for the implementation: how do we sample $p_i$ in line 3, and how do we determine its guide in line 4? We discuss several variations together with the implications on the expected running time.

IBR-Construct($\mathrm{DT}^\ell(Q_i), P_i, q$)

1 **if** $|P_i| = 4$
2 **then return** CreateQConflictDT($P_i$);
3 Sample $p_i \in P_i$;
4 guide($p_i$) $\leftarrow$ one neighbor of $p_i$ in $\mathrm{DT}^\ell(Q_i)$;
5 $P_{i-1} \leftarrow P_i \setminus \{p_i\}$;
6 $\mathrm{DT}^\ell(Q_{i-1}) \leftarrow$ LinkDTDelete($\mathrm{DT}^\ell(Q_i), p_i$);
7 $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_{i-1}) \leftarrow$ IBR-Construct($\mathrm{DT}^\ell(Q_{i-1}, P_{i-1}, q)$);
8 $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i) \leftarrow$ QConflictDTInsert($\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_{i-1}), p_i, \mathrm{guide}(p_i)$);
9 **return** $\mathrm{DT}^{\otimes}_{\mathrm{q}}(P_i)$;

*Uniform sampling.* A natural approach is to take $p_i$ uniformly at random. However, if the guide is a neighbor of $p_i$, it is not clear how to bound the running time. If the triangulations store only incidence relations, the point location time for $p_i$ will be proportional to $\mathrm{d}^\circ(\mathrm{guide}(p_i), \mathrm{DT}^\ell(Q_{i-1}))$. If $\mathrm{DT}^\ell(Q_{i-1})$ were a planar triangulation, choosing the nearest neighbor of $p_i$ as guide would yield constant

expected point location time [11, Lemma 1]. Unfortunately, as $\mathrm{DT}^\ell(Q_{i-1})$ is embedded in 3D, the nearest neighbor is no longer guaranteed to be a neighbor in $\mathrm{DT}^\ell(Q_{i-1})$.

An alternative is to use a triangle as a guide instead of a vertex. Let $\mathrm{guide}(p_i) = t_i^\ell = p_j p_k p_l$ be a triangle created by the removal of $p_i$ in $\mathrm{DT}^\ell(P_i)$ (without loss of generality $j > k, l$). Then, $t_i^\ell$ can be matched to a tetrahedron $t_i^\otimes = p_j p_k p_l q_\partial$ created in $\mathrm{DT}_\mathrm{q}^\otimes$ when $p_j$ is inserted. This matching can be efficiently computed by storing in $p_j$ all its incident triangles in $\mathrm{DT}^\ell(P_j)$ in counterclockwise order when $p_j$ is deleted in $\mathrm{DT}^\ell$. After the insertion of $p_j$ in $\mathrm{DT}_\mathrm{q}^\otimes$, we walk simultaneously around the edge $p_j q_\partial$ of $\mathrm{DT}_\mathrm{q}^\otimes(P_j)$ and through the list of triangles in order to put pointers from each $t_i^\ell$ to the matching $t_i^\otimes$. In a more powerful model of computation, we could also represent triangles as triples of indices in $[1, d]$ and maintain the correspondence between $t_i^\ell$ and $t_i^\otimes$ using universal hashing [4] in $O(1)$ expected time.

*Low-degree vertex sampling.* Instead of sampling $p_i$ at random, another choice can be to sample it uniformly among the points with degree at most 7. Since $\mathrm{DT}^\ell(Q_i)$ is planar, there are $\frac{2}{5}i$ candidates to choose from. The advantage is that we can delete $p_i$ quickly using "low degree optimization". As previously, we need to take triangles as guide. Unfortunately, $P_i$ is no longer a random subset of $P$ of size $i$, and we cannot bound the expected structural change with such a sequence.

*Low-degree edge sampling.* As already pointed out, for vertex guides to be efficient, we need to control their degrees. To this aim, instead of choosing $p_i$ first and then $\mathrm{guide}(p_i)$ amongst its neighbors, we will choose directly the edge $p_i\,\mathrm{guide}(p_i)$. The following lemma ensures that we can find an edge with $\mathrm{d}^\circ(p_i, \mathrm{DT}^\ell(Q_i)) \leq 8$, $\mathrm{d}^\circ(\mathrm{guide}(p_i), \mathrm{DT}^\ell(Q_i)) \leq 230$, and $p_i$ sampled at random in a subset of $P_i$ of size greater than $\frac{i}{96}$. As for low-degree vertex sampling, we cannot guarantee a bound on the structural change for such a permutation.

**Lemma 2.** *Let $T$ be a planar triangulation with $n$ vertices such that the external face of $T$ is a triangle. Then $T$ contains $\Omega(n)$ edges whose both endpoints have degree $O(1)$.*

*Proof.* It is well known that $T$ contains an independent set $I \subseteq P$ of vertices such that (i) $|I| \geq \frac{n}{18}$, and (ii) $\mathrm{d}^\circ(p, T) \leq 8$ for all $p \in I$. Let $N$ be the neighborhood of $I$ in $T$. The set $N$ induces a planar graph with at least $\frac{n}{18}$ facets (each facet contains a single point of $I$). Hence, $|N| \geq 2 + \frac{n}{36}$ and the average degree of a vertex in $N$ (wrt $T$) is at most $3+3\cdot 36 = 111$ (using the fact that $\forall v, \mathrm{d}^\circ(v, T) \geq 3$ if $n \geq 4$), so at least half of the vertices in $N$ have degree at most 222. Thus, at least $\frac{n}{96}$ points in $I$ have a neighbor of degree at most 230. □

**BRIO sampling.** The *BRIO*-approach [2] uses a *gradation* to construct the permutation $\{p_i\}$. We construct a sequence $P = S_r \supseteq S_{r-1} \supseteq \cdots \supseteq S_0 = \emptyset$ of subsets such that $S_{i-1}$ is obtained from $S_i$ by sampling each point independently

with probability 1/2. Note that for $p \in P$, we have $\Pr[p \in S_i] = 2^{-(r-i)}$, so $r = O(\log d)$ with high probability. Now the algorithm proceeds in $r$ *rounds*: in round $r - i + 1$, we have $\mathrm{DT}^\ell(S_i \cup \{q\})$, and we compute a spanning tree $T_i$ for $\mathrm{DT}^\ell(S_i \cup \{q\})$ that has maximum degree 3. This takes time $O(|S_i|)$, using an algorithm by Choi [10]. We store $T_i$ as a guide. Then, we delete all points from $S_i \setminus S_{i-1}$ to obtain $\mathrm{DT}^\ell(S_{i-1} \cup \{q\})$ and proceed to round $r - i + 2$. This deletion takes time $O(|S_i|)$ [6]. The construction of $\mathrm{DT}^\otimes_\mathrm{q}(P)$ also proceeds in $r$ rounds. In round $i$, we have $\mathrm{DT}^\otimes_\mathrm{q}(S_i)$, and we would like to obtain $\mathrm{DT}^\otimes_\mathrm{q}(S_{i+1})$. For this, we perform a BFS along $T_{i+1}$, starting from a vertex in $S_i$, and we insert the points as they are encountered. Since each edge of $T_{i+1}$ must appear in in $\mathrm{DT}^\otimes_\mathrm{q}(S_{i+1})$, the time for walking along each edge can be charged to the structural change. However, we need to bound the time it takes to locate the tetrahedron that is intersected by the next edge. This is done in the following lemma.

**Lemma 3.** *The total time for locating the tetrahedra that are intersected by the edges of the bounded-degree spanning tree is $O(|S_i| + C^\otimes(S_i))$.*

*Proof.* The point location takes place on the boundary of $\mathrm{DT}^\otimes_\mathrm{q}(S_i)$. The standard BRIO analysis shows that biasing the permutation in each round increases the expected structural change by at most a constant factor [2]. Thus, throughout the round the total number of triangles that can appear on the boundary are $O(|S_i| + C^\otimes(S_i))$ (the ones present initially, plus the ones created). Since $T_i$ has bounded degree, we scan each triangle at most $O(1)$ times for each incident vertex. The claim follows. □

We can summarize these results in the following theorem:

**Theorem 4.** *Backward Reinsertion takes $O(C^\otimes(P))$ expected time using*
*(i) uniform sampling using triangle guides, or*
*(ii) BRIO sampling using vertex guides*

## 5 Experimental Setup and Results

| Variant | Sampling | Guide |
|---|---|---|
| IBR-Hashing | deg $p_i \le 7$, low deg optimized delete | hash triangle |
| IBR-Neighbor | deg $p_i \le 7$, low deg optimized delete | lowest degree neighbor of $p_i$ |
| IBR-Edge | Random edge $\overline{uv}$ with $\mathrm{d}^\circ(u) \le 7, \mathrm{d}^\circ(u) + \mathrm{d}^\circ(v) \le 15$ | guide$(u) = v$ |
| IBR-BRIO | Independent rounds with probability 1/2 | BDST of Choi [10] |
| Guide-only | Constructing $\mathrm{DT}(P)$ and sew, edge guide and edge sampling. | |
| $\mathrm{DT}^\otimes_\mathrm{q}$-only | Constructing $\mathrm{DT}^\otimes_\mathrm{q}(P)$ and sew, no guide, random order. | |
| CGAL | CGAL: constructing $\mathrm{DT}(P)$ and sew, no guide, "Dijkstra order" from $\mathrm{DT}^\ell(Q)$. | |
| CGAL-rnd | Randomized CGAL: constructing $\mathrm{DT}(P)$ and sew, no guide, random order. | |

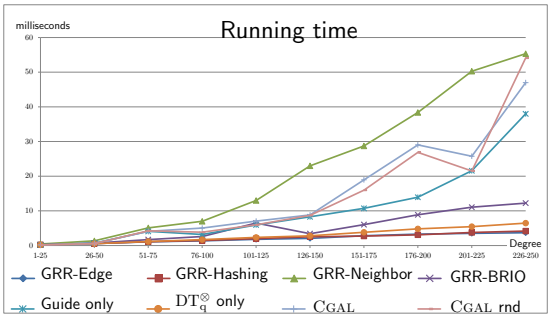
We implemented the above variants of our algorithm using CGAL 4.2.[6] In practice, our implementations differ a bit from theory. For IBR-hashing, we did

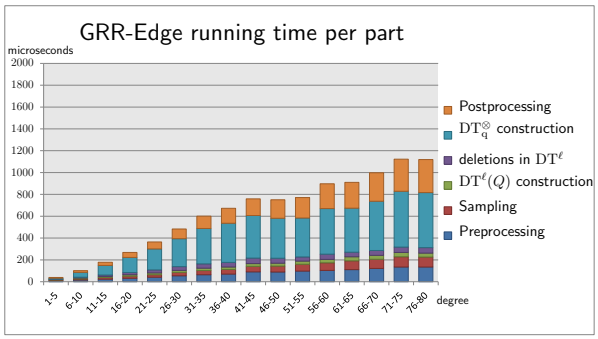
[6] The experiments were performed on a 32-bit 2.53 GHz quad-core Intel i5 running Microsoft Windows 7 operating system with 3 gigabytes usable RAM. Code has been compiled using Microsoft Visual C++ in CGAL release mode.

not use a real hash table, but a balanced binary search tree. IBR-Neighbor has not been proven to be optimal in theory, taking the neighbor of $p_i$ of lowest degree has the disadvantage of needing the computations of these degrees.

The conditions proved in Lemma 2 are too restrictive to implement IBR-Edge, thus we are looking for edges $\overline{uv}$ such that the sum of the degrees of the two end-points is less than 15. The vertex of smallest degree of such an edge is chosen as $p_i$ and the other as guide. The fact that there are $\Omega(n)$ such edges is not guaranteed by our lemma but works well in practice.

We experimented on various "reasonable" datasets (several random distributions, real 3D models) where the degree of points is bounded, and we observed similar running times for all methods. Our experimental results are interesting when we use distributions with high degree points such as points on the moment curve $\gamma(t) = (t, t^2, t^3)$ ($\mathrm{d}^\circ(v) = \Theta(n)$) and the *helix* distribution [14] ($\mathrm{d}^\circ(v) = \Theta(\sqrt{n})$). On the side picture, we show the average running time per degree for degrees up to 250 on a scale in milliseconds. The data is aggregated over all distributions, however the long term behavior is dominated by the moment curve distribution. IBR-Edge is the best variant, the good performance of $\mathrm{DT}_{\mathrm{q}}^{\otimes}$-only indicates that computing $\mathrm{DT}_{\mathrm{q}}^{\otimes}(P)$ instead of $\mathrm{DT}(P)$ is the source of a big part of the improvement, while the improvement due to the use of guides to speed up point location is less crucial. Timings above address a complete deletion, the side figure details how this time is split in the different parts of the algorithm. More details about experiments can be found in Schrijvers's thesis [17].




In our analysis of the algorithm, we have made the general position assumption. In combination with numerical stability issues, this generally does not hold in real-world data sets. Our implementation is meant as a proof-of-concept, not production-quality code. Whenever we were unable to complete a deletion because of stability issues, we have discarded the results. This has only happened a few dozen times for the 182,051 data points we have gathered. Since our algorithm works in arbitrary dimensions, it would be possible to go to a lower-dimensional DT whenever the general position assumption is violated, as CGAL currently does for their deletion code.

### Conclusion

Our vertex deletion algorithm has running time $O(C^{\otimes}(P))$, improving in theory on previous algorithms in the common circumstance that $C^{\otimes}(P) = o(k \log d)$ where $k$ is the number of tetrahedra needed to retriangulate the cavity and $d$ its number of vertices. In practice our implementation outperforms the current CGAL implementation when the deleted point has high degree ($\geq$ 100) and remains competitive for low degree. Going from theory to practice required some compromises; our best implementations differ from the theoretical model: IBR-edge uses different degrees in the sampling method while IBR-hash does not actually use hashing but a binary search tree.

The low degree sampling schemes has not been proven to have theoretically optimal complexity. It is an open question to prove or disprove that such a permutation, where the next point is randomly chosen in a linear size subset, is random enough to obtain an expected complexity $O(C^{\otimes}(P))$.

## References


doi [1] A. Aggarwal, L. Guibas, J. Saxe, & P. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discr. Comp. Geom.*, 4:591–604, 1989.

doi [2] N. Amenta, S. Choi, & G. Rote. Incremental constructions con BRIO. In *19th Sympos. Comput. Geom.*, 211–219, 2003.

doi [3] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, & M. Yvinec. Triangulations in CGAL. *Comput. Geom.*, 22:5–19, 2002.

doi [4] L. Carter & M. N. Wegman. Universal classes of hash functions. *J. Comput. System Sci.*, 18(2):143–154, 1979.

url [5] CGAL. Computational Geometry Algorithms Library, 2013. cgal.org.

doi [6] B. Chazelle & W. Mulzer. Computing hereditary convex structures. *Discr. Comp. Geom.*, 45(4):796–823, 2011.

[7] S.-W. Cheng, T. K. Dey, & J. R. Shewchuk. *Delaunay Mesh Generation*. CRC Press, Boca Raton, Florida, December 2012.

url [8] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dartmouth College, 1990.

doi [9] L. P. Chew. Guaranteed-Quality Mesh Generation for Curved Surfaces. In *9th Sympos. Computat. Geom.*, 274–280, 1993.

doi [10] S. Choi. The Delaunay tetrahedralization from Delaunay triangulated surfaces. In *18th Sympos. Comput. Geom.*, 145–150, 2002.

doi [11] O. Devillers. The Delaunay hierarchy. *Int. J. Found. Comp. Sc.*, 13:163–180, 2002.

doi [12] O. Devillers. On deletion in Delaunay triangulations. *Internat. J. Comput. Geom. Appl.*, 12(3):193–205, 2002.

doi [13] O. Devillers. Vertex removal in two-dimensional Delaunay triangulation: Speed-up by low degrees optimization. *Comput. Geom.*, 44(3):169–177, 2011.

doi [14] J. Erickson. Nice point sets can have nasty Delaunay triangulations. *Discrete and Computational Geometry*, 30(1):109–132, 2003.

doi [15] B. Joe. Three-Dimensional Triangulations from Local Transformations. *SIAM Journal on Scientific and Statistical Computing*, 10:718–741, 1989.

doi [16] D.-T. Lee & A. K. Lin. Generalized Delaunay Triangulations for Planar Graphs. *Discrete & Computational Geometry*, 1:201–217, 1986.

url [17] O. Schrijvers. Insertions and deletions in Delaunay triangulations using guided point location. Master's thesis, Technische Universiteit Eindhoven, 2012.