On Deletion in Delaunay Triangulations

Olivier Devillers

published in International Journal of Computational Geometry & Applications, 3:429-442, 1993

On Deletion in Delaunay Triangulations

Olivier Devillers

Abstract

This paper presents how the space of spheres and shelling may be used to delete a point from a d-dimensional triangulation efficiently. In dimension two, if k is the degree of the deleted vertex, the complexity is $O(k \log k)$, but we notice that this number only applies to low cost operations, while time consuming computations are only done a linear number of times.

This algorithm may be viewed as a variation of Heller's algorithm,[1, 2] which is popular in the geographic information system community. Unfortunately, Heller algorithm is false, as explained in this paper.

keywords: Computational geometry, Delaunay, power, space of circles

1 Introduction

The computation of the Delaunay triangulation of a set S of n points in the plane is one of the classical problems of computational geometry.

Many structures and algorithms have been proposed in the past to compute Delaunay triangulations. Some of these algorithms have the two following properties: they are incremental and they do not used complicated data structures in addition to the triangulation itself. Among these algorithms, let us mention the historical algorithm of Green and Sibson [3], or some other variants [4, 5, 6, 7, 8]. All perform a walk in the triangulation to accelerate point location.

The advantage of that category of incremental Delaunay algorithms is that they may easily be turned into fully dynamic Delaunay algorithms. Since there is no complicated data structure for point location, the deletion of a point is reduced to the deletion in the triangulation itself.

Definition and notations

Given a set $\mathcal{S}$ of points in d-dimensional space, $\mathcal{DT}(\mathcal{S})$, the Delaunay triangulation of $\mathcal{S}$ is defined by the following property: $d+1$ *points of* $\mathcal{S}$ *are the vertices of a Delaunay simplex if and only if the sphere passing through these points does not contain another point of* $\mathcal{S}$ *in its interior* (see Figure 1 for a Delaunay triangulation in two dimensions).

[0]INRIA, BP 93, 06902 Sophia Antipolis cedex, France. E-mail: Firstname.Lastname@sophia.inria.fr . This work was partially supported by ESPRIT LTR 21957 (CGAL).

Given the Delaunay triangulation $\mathcal{DT}(\mathcal{S})$ and a vertex p in $\mathcal{DT}(\mathcal{S})$, we address the problem of finding $\mathcal{DT}(\mathcal{S} \setminus \{p\})$.

In two dimensions, the natural parameter to evaluate the complexity of this problem is the degree k of p in $\mathcal{DT}(\mathcal{S})$, since the deletion of p means that k triangles must be removed from the triangulation and $k-2$ new triangles must be created to fill this hole. In the worst case, k may be $|\mathcal{S}|$, but if p is chosen randomly in $\mathcal{S}$, then it is well known that the expected value of k is 6, without any assumption on the point distribution.

In higher dimensions, the number of simplices incident to p is not directly related to the number of simplices created to fill the hole. We will let f denote the sum of these two numbers. In the worst case, the whole Delaunay triangulation may be affected, and $f = O(n^{\lfloor \frac{d+1}{2} \rfloor})$. This distribution does not reflect practical configurations, and a constant value of f is more likely in practice. For a uniform point distribution, the expected value of f may be shown to be constant. In three dimensions, the expected number of deleted tetrahedron for Poisson distribution is $\frac{96}{35}\pi^2 \simeq 27$ [9].

Thus, even if we do not want to neglect the possible case of a big value for k, we have to keep in mind that a good algorithm must perform well on small values of k.

Previous related work

Classical Computational Geometry has already addressed the problem of deleting points from Delaunay triangulations. This can be done with optimal asymptotic complexity $O(k)$ in 2 dimensions [10, 11]. But these algorithms are a little bit too intricate and the big O hides too important a constant for them to be good algorithms for reasonable values of k (we give some details in Section 4.1.3 for Chew's algorithm, Aggarwal et al's algorithm is even more complicated).

Practitioners often prefer algorithmic simplicity to theoretical optimality, and favour a simple suboptimal $O(k^2)$, implementation of the deletion algorithm. This may be achieved, for example, by flipping, to reduce the degree of the deleted vertex to 3, and flipping again to restore the Delaunay property. Another simple algorithm consists in finding the Delaunay triangle incident to an edge of the hole in $O(k)$ time, which also yields an $O(k^2)$ time algorithm.

A very simple $O(k \log k)$ solution was suggested by Heller [1, 2], in which successive ears are filled in turn. In that way, during the algorithm we always have a simple polygon of decreasing size to triangulate. Unfortunately, this solution is wrong, but we will show in this paper how to correct it.

Overview

In this paper, we provide a very simple and efficient $O(k \log k)$ algorithm to delete a vertex in a planar Delaunay triangulation based on shelling [12, 13] and duality [14, 15]. We also discuss the effective complexity of this algorithm and of a few others for small values of k. We will study the different kinds of geometric predicates necessary for these algorithms.

This algorithm generalizes well in higher dimensions: its time complexity becomes $O(f \log f)$, where f is the number of tetrahedra created, and it also

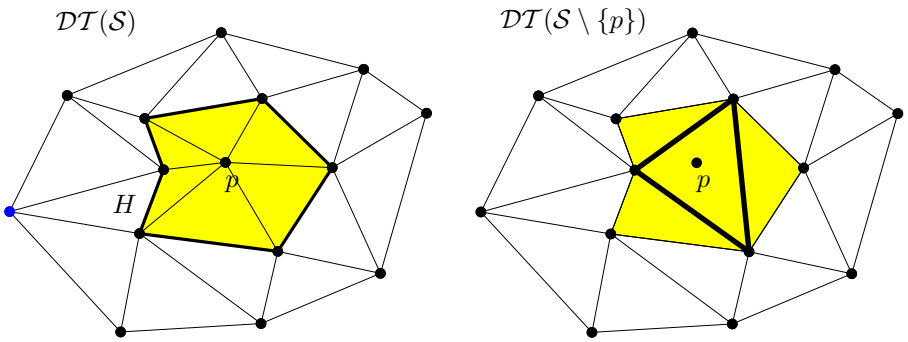

Figure 1: Deletion of a vertex.

generalizes to regular triangulations (power diagrams). Of course this deletion algorithm apply to regular triangulation only if all the points belong to the triangulation, discovering hidden points which reappear when a point is deleted is a more difficult task.

2 Two Dimensional Algorithm

A deletion algorithm has to remove all triangles incident to p and retriangulate "Delaunay-wise" the star-shaped polygon $H = \{q_0, q_1, \ldots, q_{k-1}, q_k = q_0\}$ created by these removals (see Figure 1).

2.1 Ears, and a Wrong Algorithm

We first define what an ear of a polygon is. Three consecutive vertices $q_i q_{i+1} q_{i+2}$ along H's boundary are said to form an ear of H if the line segment $q_i q_{i+2}$ is inside H and does not cross its boundary. An ear of H is said to be Delaunay, if the circle through q_i q_{i+1} and q_{i+2} does not contain any other vertices of H in its interior. Heller [1] (also cited by Midtbø [2]) claimed (without proof) that among all the potential ears $q_i q_{i+1} q_{i+2}$ of H, the one having the circumcircle with smallest radius is a Delaunay ear. This claim is false, as illustrated by Figure 2: on the left handside are shown the Delaunay triangulation $\mathcal{DT}(\mathcal{S})$ and the hole H to be retriangulated; on the right handside are shown two potential ears $q_0 q_1 q_2$ and $q_1 q_2 q_3$. $q_0 q_1 q_2$ has the smallest circumcircle among all ears of H, but it contains q_3, which invalidates Heller's claim. Heller's mistake is to assume that when we deform a circle through $q_1 q_2$ to maximise its portion inside H, the radius increases, but this is true only if the center of the circle is inside H, which is not the case in Figure 2.

In fact, the idea of finding an ear belonging to the Delaunay triangulation works, but with another criterion, as explained below.

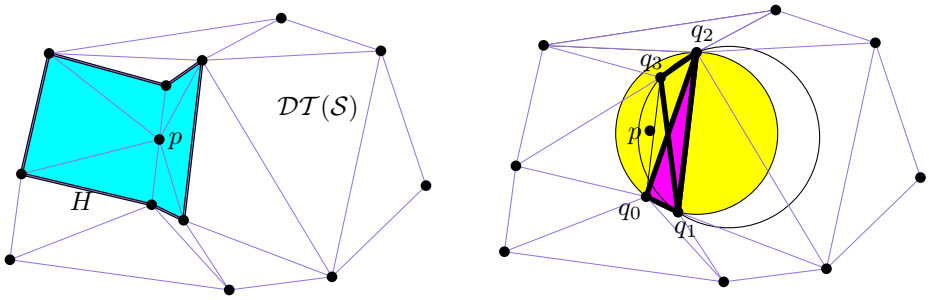

Figure 2: The smallest potential ear may not belong to the final Delaunay triangulation. The shaded triangle is the ear with smallest circumradius, but its associated circle contains q_3 and thus is not a Delaunay ear.

2.2 Delaunay and Convex Hull

There exists a well known duality between Delaunay triangulations in dimension d and convex hulls in dimension $d+1$ [16, 17, 14]. If we associate to a point $p = (x, y) \in \mathcal{S}$ a point $p^\star = (x, y, x^2 + y^2)$ on the paraboloid Π of equation $z = x^2 + y^2$, the Delaunay triangulation of $\mathcal{S}$ is the projection of the convex hull of the 3D points. The reason is that for $p, q, r, s \in \mathcal{S}$, p is inside the circle C_{qrs} through qrs if and only if $p^\star$ is below the plane P_{qrs} through $q^\star r^\star s^\star$ ($\Pi \cap P_{qrs}$ projects onto circle C_{qrs}). We even have the equality between the power of p with respect to C_{qrs} and the signed vertical distance between $p^\star$ and P_{qrs}.[1]

2.3 Shelling

Convex hulls may be computed by the shelling algorithm [13]. The shelling [12] of a convex polyhedron P is the enumeration of its faces in some appropriate order. Imagine an observer is moving along a line l going through the polyhedron, starting at the intersection of P and l. At the starting position, the observer can only see one face of P (the face intersecting its trajectory) and when she moves away from P she discovers other faces one by one; at infinity, the observer sees "half" of P. Then the observer turns round at infinity on the opposite side of l (where she sees the other half of P), and then moves on l, enumerating the faces of P when they disappear from her view. This order is called the shelling order of P with respect to l; it has the property that the set of enumerated faces remains simply connected during the enumeration. Seidel's algorithm for convex hull reports the faces of the convex hulls in that order,

[1] If C is a circle of center x and radius r, p is a point and l is a line through p intersecting C in t and u, then $power(p, C) = |xp|^2 - r^2 = \overline{pt}\,\overline{pu}$ where $\overline{yz}$ is the signed length of yz. $power(p, C)$ is zero on C boundary, negative inside and positive outside. When C is given by three points, the power may be known without computing the circumradius, as explained in Section 3.1. Power is negative inside the circle and positive outside.

by maintaining a priority queue of potential new faces. These new faces are of two kinds depending on whether the next face contains a new vertex, or else connects already visible vertices.

2.4 Deletion in Delaunay

Let us now use the idea of finding Delaunay ears to retriangulate the hole created by the deletion of a point p in $\mathcal{DT}(\mathcal{S})$. Using duality with convex hulls, the problem is transformed into filling the hole in the convex hull created by the deletion of $p^\star$. Then we may note that if we use shelling order with respect to the vertical line through $p^\star$, an observer (going up) reaching $p^\star$ sees the boundary of this hole exactly; thus, the end of the shelling procedure corresponds exactly to the triangulation of the hole. This partial shelling is easier to implement than Seidel's algorithm, since all the vertices are already visible and thus only one kind of potential new faces has to be found. The already noted correspondence between vertical distance and power yields the following lemma.

> **Lemma:** *Consider a polygon* $H = \{q_0, q_1, \ldots, q_{k-1}, q_k = q_0\}$ *and a point* p *such that the edges of* q_iq_{i+1} *belongs to the Delaunay triangulation of* $\{q_0, q_1, \ldots, q_{k-1}, p\}$. *If* $|power(p, circle(q_i, q_{i+1}, q_{i+2}))|$ *is maximal, then* q_iq_{i+2} *is an edge of the Delaunay triangulation of* $\{q_0, q_1, \ldots, q_{k-1}\}$

Thus the deletion of a point may be implemented in a simple way, by maintaining a structure to store the ears. The ears are naturally ordered along the boundary of the hole, each with its priority. This structure must support the following operations : find next and previous ear according to counterclockwise order along H's boundary, delete ear with minimum priority and update the priority of an ear. This structure may be implemented with any dictionary structure augmented by next and previous pointers.

Algorithm *Delete*$(\mathcal{DT}(\mathcal{S}), p))$

```
1.  Let q_0 q_1 ... q_{k-1} be the vertices incident to p in DT(S) in ccw order around
    p;
2.  Let Q be a priority queue;
3.  for i = 0 to k - 1
4.      do ear ←q_i q_{i+1} q_{i+2};
5.          if clockwise(q_i q_{i+1} q_{i+2})
6.             then p ←∞;                                   //not an ear
7.             else  p ←−power(p, ear);            //inside circle power< 0
8.          Q.insert(p,ear);                          //insert(priority,key)
9.  while Q.size()> 3
10.    do ear ←Q.minimum();
11.       create triangle ear and link it to its two existing neighbors;
12.       ear0 ←ear.previous;
13.       ear1 ←ear.next;
14.       ear0.vertex(2) ←ear.vertex(2); ear0.next ←ear1;
```

15. $ear1$.vertex(0) ←ear.vertex(0); $ear1$.previous ←$ear2$;
16. Q.delete(ear);
17. Q.modify-priority($ear0$);
18. Q.modify-priority($ear1$);
19. ear ←Q.minimum(); *//the three last ears are identical*
20. create triangle ear and link it to its three existing neighbors;

Higher dimensions The generalization to d dimensions is easy. The boundary of the region to retriangulate is a simple polyhedron H, and the ears are simplices formed by the vertices of two incident facets of H. The difference is that the same simplex may correspond to $O(d^2)$ pairs of incident facets, and that the creation of an ear may modify $O(d^2)$ other ears in the priority queue.

3 2D Analysis

3.1 Power Computation

An analytical expression of the power of p with respect to $q_0q_1q_2$ is

$$power(p, circle(q_0, q_1, q_2)) = \frac{\begin{vmatrix} x_{q_0} & x_{q_1} & x_{q_2} & x_p \\ y_{q_0} & y_{q_1} & y_{q_2} & y_p \\ x_{q_0}^2 + y_{q_0}^2 & x_{q_1}^2 + y_{q_1}^2 & x_{q_2}^2 + y_{q_2}^2 & x_p^2 + y_p^2 \\ 1 & 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} x_{q_0} & x_{q_1} & x_{q_2} \\ y_{q_0} & y_{q_1} & y_{q_2} \\ 1 & 1 & 1 \end{vmatrix}}$$

The 3×3 determinant is the orientation test of $q_0q_1q_2$, and the 4×4 determinant the incircle test of p with respect to $q_0q_1q_2$. First note that if $q_0q_1q_2$ has the wrong orientation, then it is not an ear and thus the 4×4 determinant does not need to be computed, and also that the orientation test is a minor of the incircle test, and thus the power computation just requires one division in addition to the usual incircle test.

Using a dynamic programming development of the determinant, the power computation requires 14 additions, 15 multiplications and one division.

Finally, note that if only q_2 changes, we do not need to recompute everything. In fact, we can do it with 6 additions, 7 multiplications and one division.

3.2 Complexity

The theoretical asymptotic complexity of this algorithm is clearly $O(k \log k)$, but this complexity only concerns the management of the priority queue, which involves relatively cheap operations (pointer manipulations and comparisons of computed powers).

Since the most expensive geometric operations are the power computations, we shall count the number of such operations exactly. The initial size of the

priority queue is k, and thus its initialization requires at most k power computations. Each ear creation implies the modification of two other ears, and thus two powers must be recomputed, and the deletion is completed when the size of the queue is 3; thus the total number of power computations is $k+2(k-4) = 3k-8$.

It is possible, as noticed above, to update the power of p with respect to $q_iq_{i+1}q_{i+2}$ when ear $q_{i+1}q_{i+2}q_{i+3}$ is processed. In the new polygon $H \setminus \{q_{i+2}\}$, $q_iq_{i+1}q_{i+3}$ is an ear and the power of p with respect to $q_iq_{i+1}q_{i+3}$ may be obtained by updating the power of p with respect to $q_iq_{i+1}q_{i+2}$ at a cheaper cost. Then the total number of computations becomes $2k-4$ power computations and $k-4$ power updates.

3.3 Robustness Issues and Degeneracies

The algorithm presented above does not address robustness issues. If floating point arithmetic is used to perform power computations, the results are rounded and their comparisons could be evaluated erroneously. We can first observe that the deletion algorithm will terminate even with incorrect arithmetic: it fill ears in turn and thus constructs a topological triangulation, which may be non Delaunay, or even have a non-planar embedding. But, even if producing a non-exact Delaunay triangulation may be acceptable for the deletion algorithm, it is unacceptable for many insertion algorithms which are not capable of processing non-exact triangulations.

The usual way to solve robustness issues consists in using exact arithmetic. To ensure good performance, we can use arithmetic filters to use exact computations in power comparisons only in difficult cases where the two powers are close. Exact computations are then used to take the right decision. This approach of filtering out easy cases has been proven efficient on the orientation and incircle tests [18, 19]. Degeneracies can be solved using perturbation techniques [20, 21].

4 Alternative Methods and Practical Results

4.1 Alternative Methods in Two Dimensions

4.1.1 Diagonal flipping

One of the interests of a method using diagonal flipping is that it does not introduce new geometric predicates: it only uses incircle tests, and thus has lower degree and generalizes easily to various metrics. However such a method may require $O(k^2)$ incircle tests in the worst case, and is not so simple to code efficiently. A good implementation of a flipping method [22] will retriangulate H by basically linking all q_i to q_0 and flipping the edges turning around q_0. In the worst case, such a method may use $(k-3)+(k-4)+\ldots+1 = \frac{(k-2)(k-3)}{2}$ incircle tests.

With this flipping method, the triangulation may be not planar at some intermediate stage although planarity is restored at the end. Because of that

coding this algorithm and particularly, traversing a non planar triangulation can be a little bit tricky. A simpler solution may consist in maintaining a queue of edges to be tested for potential flip. Each time the diagonal of a quadrilateral is flipped, the four edges of the quadrilateral are inserted in the queue. This simpler algorithm will make much more incircle tests, since the flip are not performed in some relevant order as above.

4.1.2 Edge completion

A second method consists in finding q_i such that $q_0q_1q_i$ is a Delaunay triangle, which may be done in $k-3$ incircle tests, and triangulating recursively the two holes. In the worst case, the number of incircle tests is exactly the same that in the flipping method.

4.1.3 Randomized algorithm

Chew [10] randomized algorithm compute the Delaunay triangulation of a convex polygon, but can apply also to the triangulation of the hole $H = \{q_0, q_1, \ldots, q_{k-1}, q_k = q_0\}$ created by the deletion of p in $\mathcal{DT}(\mathcal{S})$. This algorithm chose at random a point q_j, triangulate recursively the interior of polygon $H' = \{q_0, q_1, \ldots, q_{j-1}, q_{j+1}, \ldots, q_{k-1}, q_k\}$ and then insert the points q_j; by construction we know that the edge q_{j-1}, q_{j+1} belongs to $\mathcal{DT}(H')$ and that q_j is incident to q_{j-1} and q_{j+1} in $\mathcal{DT}(H)$, thus inserting q_j can be done without locating q_j first, since the expected number of triangles having q_j as vertex in $DT(H)$ is 3 we get a constant complexity to insert q_j and a linear complexity overall. More precisely, when q_j is inserted, incircle tests must be performed for the triangles of $\mathcal{DT}(H')$ destroyed by the insertion (2 on average) and their neighbors inside H' (between 2 and 3 on average) which yields a total of $5k+O(1)$ incircle tests.

Implementation of Chew algorithm needs to be done with some special auxiliary data structure to manage the triangulation of a simple polygon and then sew with the hole H since recursive steps such as triangulation of H' the boundary of the polygon H' does not necessary belongs to $\mathcal{DT}(\mathcal{S})$.

4.1.4 On alternative methods

For small value of $k < 9$, flipping or edge completion requires less incircle tests computations, but the simplicity of our "ear-queue" algorithm and its good performance for $k \geq 9$ make it a very good candidate for Delaunay vertex deletion. Nevertheless, it can be interesting to treat as special cases some small k values such as $k = 4$ and $k = 5$.

4.2 Higher Dimensions

In higher dimensions, flipping, edge completion and shelling algorithms generalize but things became more difficult. The flipping must be done in a higher dimension, which makes it more intricate to implement [23]. Edge completion transforms into facet completion, and must deal with the triangulation of non

simply connected polyhedra. Shelling is the easiest method to generalize. Furthermore, the increase in the average value of k with the dimension reinforces its advantage over alternative candidates in higher dimensions.

4.3 Experimental Results

4.3.1 Code and data

This algorithm was implemented within the author's simple hierarchical structure [6].

Robustness issues are solved using 24 bits integers to store points coordinates. Geometric predicates are computed with approximate arithmetic and the exactness of the result can be ensured by static and semi-static filters[2]. The filters failure are backed up by exact computations.

According to paragraphs above, we have coded several versions of the deletion procedure.

- **ear3.** The basic version using the ear queue explained in this paper.
- **ear5.** A variant processing the ears' queue while its size is greater than five. When the region to triangulate is a pentagon, a specific method using two or three incircle tests is used . Degree three, four and five vertices are removed by a specific algorithms.
- **flip.** The flipping-based method briefly described above.
- d_{limit}. A **mixed** method using **ear5** if the degree of the removed point is $\geq d_{limit}$ and the **flip** method otherwise.

4.3.2 Results

The code was tested on a 2,000,000 points set uniformly distributed in a square (Figure 3). The Delaunay triangulation of the points is first computed (in 104 seconds). Next the points are removed in a random order. We tried the methods ear3, ear5, flip and the mixed method for $5 \leq d_{limit} \leq 11$. Figure 4 provides the whole deletion time, the number of incircle predicate evaluations and the number of power computations.

These experiments have been done on a Sun Ultra10 300MHz workstation 256Mo main memory. The code is written in C++ and compiled with AT-T's compiler with optimizing options. Times were obtained with the `clock` command and are given in seconds and are only for the deletion phase.

[2]Filters can be classified in static, semi-static and dynamic [24]:
static: error bound is determined at compile time,
dynamic: error bound is determined at run time, usually with an error computation for all intermediate results,
semi-static: error bound is mainly determined at compile time, with addition of very few computation at run time.

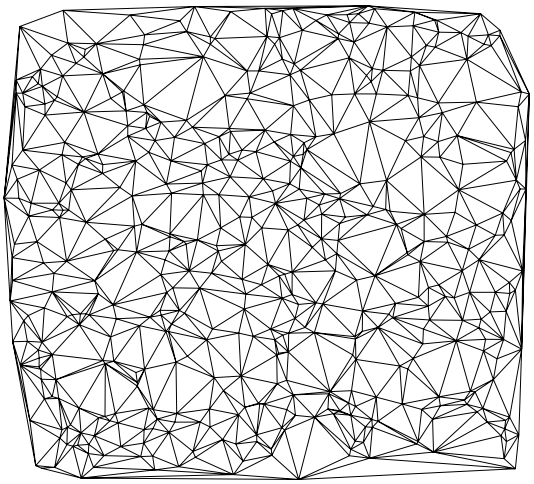

Figure 3: Delaunay triangulation of random points in a square.

As infered from the theory, the performance is optimal when d_{limit} is about 9. The mixed method therefore reduces the complexity of the deletion of high degree vertices from $O(k^2)$ to $O(k \log k)$.

5 Lower Bound

The ear-queue method, i.e. the construction of the ears in the right order, has a $\Omega(k \log k)$ lower bound. Consider the origin r and points $p_i, 0 \leq i \leq n$ and $q_i, 0 \leq i \leq n$ so that angles $p_i r q_i = q_i r p_{i+1} = \frac{\pi}{n}$ and distances $p_i r = 1$ and $q_i r = x_i, 1 < x_i < \alpha$. α is chosen so that the Delaunay triangulation of $\mathcal{S} = \{r, p_0 \dots p_n, q_0 \dots q_n\}$ links r to all other points (Figure 5). When deleting r, all $p_i q_i p_{i+1}$ are Delaunay ears, but finding the right order on this ears, which is not necessary to find the new Delaunay triangulation, is equivalent to sorting the x_i and thus as a $\Omega(n \log n)$ lower bound.

6 Conclusion

We have proposed a simple method for point deletion in Delaunay triangulations. This method guarantees an $O(k \log k)$ complexity where k is the degree of the removed point while most alternatives have a quadratic behavior in the worst case. The implementation in two dimensions corroborates these results and shows a good behavior in practice. The algorithm should be even more efficient in higher dimensions due to the lack of alternative methods and the higher average degree of a Delaunay vertex.

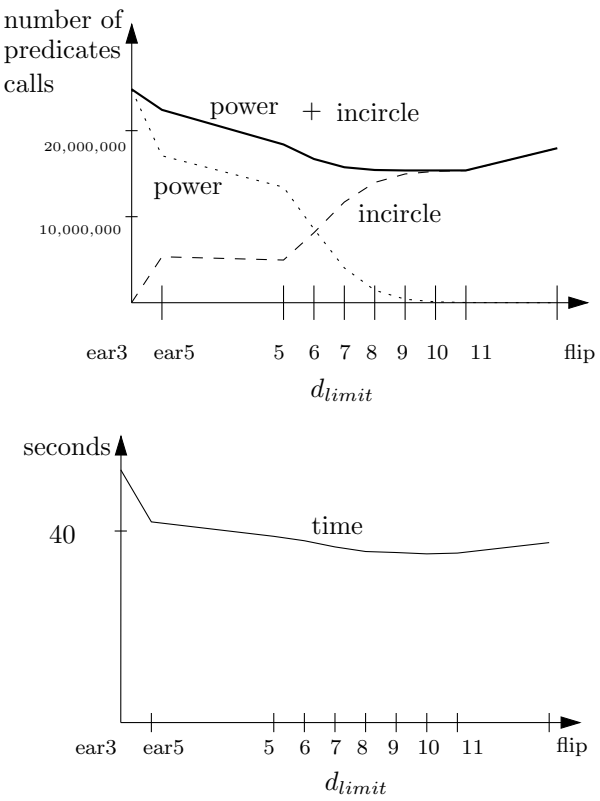

Figure 4: Deletion time for two millions of random points in a square for different deletion methods. See paragraph "Results" for details.

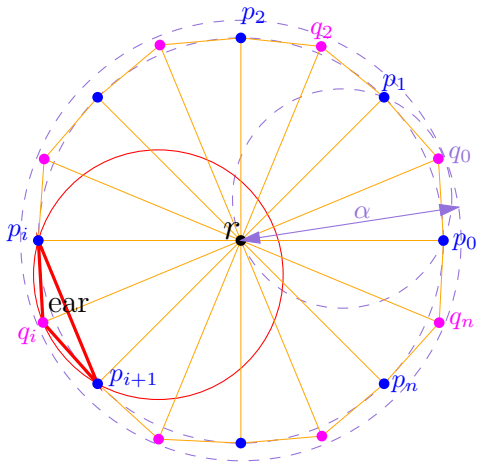

Figure 5: For $\Omega(k \log k)$ lower bound.

Code A compiled demo version is available at
http://www.inria.fr/prisme/logiciels/del-hierarchy/.

Acknowledgment The author would like to thank Jean-Michel Moreau, Jack Snoeyink and Mariette Yvinec for helpful discussions and careful reading of this paper.

References

[1] M. Heller. Triangulation algorithms for adaptive terrain modeling. In *Proc. 4th Internat. Sympos. Spatial Data Handling*, pages 163–174, 1990.

[2] T. Midtbø. *Spatial Modelling by Delaunay Networks of Two and Three Dimensions*. Ph.D. thesis, Norvegian Institute of Technology, Trondheim, 1993.

[3] P. J. Green and R. R. Sibson. Computing Dirichlet tessellations in the plane. *Comput. J.*, 21:168–173, 1978.

[4] Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1996.

[5] P. Bose and L. Devroye. Intersections with random geometric objects. *Comput. Geom. Theory Appl.*, 10:139–154, 1998.

[6] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 106–115, 1998.

[7] L. Devroye, C. Lemaire, and J.-M. Moreau. Fast delaunay point location with search structures. In *Proc. 11th Canad. Conf. Comput. Geom.*, 1999.

[8] C. Lemaire. *Triangulation de Delaunay et arbres multidimensionnels*. Thèse de doctorat en sciences, École des Mines de St-Etienne, France, 1997.

[9] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.

[10] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH, 1986.

[11] A. Aggarwal, Leonidas J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604, 1989.

[12] H. Bruggesser and P. Mani. Shellable decompositions of cells and spheres. *Math. Scand.*, 29:197–205, 1971.

[13] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proc. 18th Annu. ACM Sympos. Theory Comput.*, pages 404–413, 1986.

[14] Olivier Devillers, Stefan Meiser, and Monique Teillaud. The space of spheres, a geometric tool to unify duality results on Voronoi diagrams. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 263–268, 1992.

[15] D. Pedoe. *Geometry, a comprehensive course*. Dover Publications, New York, 1970.

[16] H. Edelsbrunner, J. O'Rourke, and R. Seidel Constructing arrangements of lines and hyperplanes with applications *SIAM J. Comput.*, 15:341–363, 1986.

[17] F. Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM J. Comput.*, 16:78–96, 1987.

[18] O. Devillers and F. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete and Computational Geometry*, 20:523–547, 1998.

[19] Olivier Devillers and Franco P. Preparata. Further results on arithmetic filters for geometric predicates, 1999. *Comput. Geom. Theory Appl*, 13:141–148, 1999.

[20] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19:1–17, 1998.

[21] P. Alliez, O. Devillers, and J. Snoeyink. Removing degeneracies by perturbing the problem or the world. Research Report 3316, INRIA, 1997.

[22] J. Snoeyink. Non redundant flip for point deletion in delaunay triangulation, 1998. Personnal communication.

[23] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[24] Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 165–174, 1998.