Delaunay triangulation of imprecise points, preprocess and actually get a fast query time

Olivier Devillers

Delaunay triangulation of imprecise points, preprocess and actually get a fast query time

Olivier Devillers
INRIA Sophia Antipolis Méditerranée

XIV Spanish Meeting on Computational Geometry, 27:30 June 2011

Abstract

We propose a new algorithm to preprocess a set of n disjoint unit disks in $O(n \log n)$ expected time, allowing to compute the Delaunay triangulation of a set of n points, one from each disk, in $O(n)$ expected time. This work reaches the same asymptotical theoretical complexities as previous results on this problem, but our algorithm is much simpler and efficient in practice.

1 Introduction

Löffler and Snoeyink [8] proposed an algorithm that preprocesses a set of disjoint unit disks in the plane in $O(n \log n)$ time and computes the Delaunay triangulation of an instance in $O(n)$ time. This algorithm has a reasonably simple description but uses as a building block the linear time construction of the constrained Delaunay triangulation of a simple polygon [4], which makes the result mainly theoretical. Buchin et al. [2] proposed a simpler solution, which uses the split of a Delaunay triangulation in linear time [3]. This solution remains a bit heavy in practice; indeed, in the preprocessing phase, they compute a Delaunay triangulation of $8n$ points (at the center and on the boundary of the disks), then the points of the instance are added in linear time to get a triangulation of $9n$ points, and finally this triangulation is split in the triangulation of the instance and the $8n$ points triangulation again. In the same paper, a different algorithm based on quadtrees is proposed allowing overlapping disks of different radii.

Contribution. In this paper we preprocess in $O(n \log n)$ randomized expected time a set of n disjoint unit disks, allowing the computation of the Delaunay triangulation of an instance taken in these disks in randomized expected $O(n)$ time. Compared to previous algorithms [8, 2] theoretical asymptotic complexity is not improved, but the proposed algorithm is much simpler, so simple that its description fits in a dozen of lines.

Moreover, the algorithm is quite efficient in practice and uses only the classical predicates for Delaunay triangulation. Our benchmarks conclude that we can process an instance much faster than with the Delaunay hierarchy [6, 10] and faster than the incremental algorithm inserting the points in spatial sorting order [5, 1].

The algorithm works for any set of circles (overlapping, different radii) and generalize to balls in higher dimensions, but to yield to a good complexity the analysis requires that the imprecise

[2]Supported by ANR grant Triangles (ANR-07-BLAN-0319).

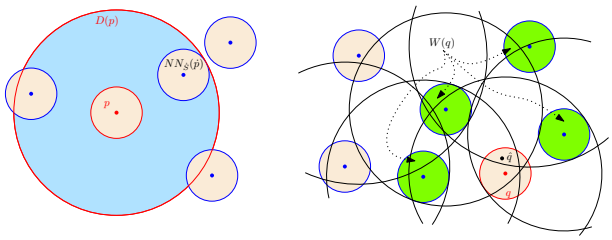

Figure 1: Definitions of $D(p)$ (big disk on the left) and $W(q)$ (darker disks on the right).

points are unit disks in the plane, possibly overlapping a constant number of times (at most k disks have a common intersection). This analysis can be extended to unit balls in higher dimensions under some suitable hypotheses. For disks of different radii overlapping at most twice, we provide a pathological example where our algorithm reaches a quadratic behavior. For disjoint disks of different radii the analysis remains open.

2 Algorithm

Notations.
• Given a point set P in the plane, let DT_P denote its Delaunay triangulation, NN_P its nearest neighbor graph, and $NN_P(v)$ the nearest neighbor of $v \in P$ in $P \setminus \{v\}$.
• For a graph G, and a vertex v of G, $d^\circ_G(v)$ is the degree of v in G.
• If p denotes an imprecise point, $\dot{p}$ denotes the center of p and $\hat{p}$ an instance of p. Let $S_k = \{p_1, p_2 \dots p_k\}$, $\dot{S}_k = \{\dot{p}_1 \dots \dot{p}_k\}$ and $\hat{S}_k = \{\hat{p}_1 \dots \hat{p}_k\}$; $S = S_n$, $\dot{S} = \dot{S}_n$ and $\hat{S} = \hat{S}_n$.
• In the case where S is a set of disjoint unit disks, given $p \in S$, we define $D(p)$ to be the disk with center $\dot{p}$ and radius $|\dot{p}NN_{\dot{S}}(\dot{p})| + 1$, that is, $D(p)$ is the interior of the circle centered at $\dot{p}$ tangent to the nearest disk in S and containing it (see Figure 1-left).
• Given an instance $\hat{S}$, we also define $W(q) = \{p \in S \setminus \{q\}; \hat{q} \in D(p)\}$ (see Figure 1-right).
• $|W|$ denotes the size of a set W.

Preprocessing. First we assume that the indices in $S = \{p_1, p_2, \dots p_n\}$ enumerate the disks in a random order (otherwise reorder the disks according to a random permutation).

We compute $DT_{\dot{S}}$ incrementally, inserting the points in the order of their indices. Furthermore after inserting $\dot{p}_k$, we compute the index $h(k)$ such that $NN_{\dot{S}_k}(\dot{p}_k) = \dot{p}_{h(k)}$. Index $h(k)$ is called the *hint* for p_k. Using incremental randomized construction, it can be done in $O(n \log n)$ expected time [6] (including the computation of $h(k)$ for every k).

Instance processing. Now given an instance $\hat{S}$, we compute $DT_{\hat{S}}$ incrementally, inserting the points in the order of their indices. The location of $\hat{p}_k$ in $DT_{\hat{S}_{k-1}}$ is done by a straight walk starting at $\hat{p}_{h(k)}$.

Complexity: *The expected cost of constructing $DT_{\hat{S}}$ is linear.*
Sketch of proof. (complete proof in [7]) By usual backward analysis, it is enough to prove that the insertion of the last point $\hat{p}$ is done in expected constant time.

Let $\hat{x}$ be the starting point of the straight walk in $DT_{\hat{S}\setminus\{\hat{p}\}}$ to insert $\hat{p}$. As seen in the algorithm description, we have $\dot{x} = NN_{\dot{S}}(\dot{p})$.

The cost of locating and inserting $\hat{p}$ is split in three parts:
- the cost c_1 of visiting the triangles incident to $\hat{x}$ in $DT_{\hat{S}\setminus\{\hat{p}\}}$ (to find the first one crossed by line segment $\hat{x}\hat{p}$),
- the cost c_2 of visiting the triangles crossed by line segment $\hat{x}\hat{p}$, and
- the cost c_3 of modifying the triangulation to update $DT_{\hat{S}\setminus\{\hat{p}\}}$ into $DT_{\hat{S}}$.

The cost c_3 is $d^\circ_{DT_{\hat{S}}}(\hat{p})$, since $\hat{P}$ is a random point in $\hat{S}$ the expected value of c_3 is less than 6.

The cost c_1 is is $d^\circ_{DT_{\hat{S}\setminus\{\hat{p}\}}}(\hat{x}) \leq d^\circ_{DT_{\hat{S}}}(\hat{x}) + d^\circ_{DT_{\hat{S}}}(\hat{p})$. $\hat{x}$ is not a random point, but it can be shown to be random enough to have an expected degree less than 36, thus the expected value of c_3 is less than 42.

The cost c_2 is related to the average size of $|W(q)|$ for $q \in S$ and one can prove that the expected value of c_2 is less than 132. ◇

3 Experiments

Algorithms proposed in previous works [8, 2] are not implemented and are rather complicated, thus we have no doubt that our solution is better in practice. The aim of this section is to compare our algorithm and a direct computation of the Delaunay triangulation of the instance by a state-of-the-art algorithm without prepocessing.

For comparison, we use Sewchuk's code: Triangle [9] and the best implementations available in CGAL [10] where the points are ordered along a space filling curve and then inserted in the *spatial sort* order [5], the point location is done by a straight walk from the previous point (point location should be fast since it starts at a point nearby).

Point sets. We experiment on random disjoint unit disks. Each set contains n disjoint imprecise points in a $4\sqrt{n} \times 4\sqrt{n}$ square. Each center of an imprecise point is generated at random in the square, then the point is kept if its distance to previously kept points is greater than 1. Points are generated until a set of n points is obtained. A point instance is generated at random in each imprecise point. Point sets with size ranging between 10^3 and 10^8 points have been generated.

Platform. Experiments have been done on the following platform:
— CGAL 3.8 (`gcc 4.3.2` release mode), — Linux-FC10,
— timings obtained with the `CGAL::Timer`, — 3.00 GHz processor 32 GByte RAM.

Results. In the following table, we report the average time for a point insertion and the average number of triangles visited during the straight walks used in point locations.

2D random imprecise points	running time (μs) per point ♯ **visited triangles per point** ♯ cache miss ⋆ per point					
n	10^3	10^4	10^5	10^6	10^7	10^8
Shewchuk (divide & conquer)	0.98 5.3	1.12 2.3	1.05 2.8	1.67 20	2.34 39	
spatial sort + insert in order	1.2 **3.74** 14	0.92 **3.63** 3.6	0.90 **3.71** 3.1	0.98 **3.67** 5.3	1.12 **3.55** 7.7	1.36 **3.71**
locate from **hint** in random order	0.9 **2.83** 14	0.81 **2.80** 3.5	1.25 **2.77** 9.7	3.1 **2.75** 27	4.1 **2.75** 32	6.1 **2.74**
locate from **hint** in spatial sort order	0.9 **2.82** 14	0.78 **2.80** 3.5	0.64 **2.77** 3.3	0.66 **2.76** 3.8	0.67 **2.75** 3.8	0.79 **2.75**

⋆ number of cache miss obtained by `valgrind - -tool=cachegrind` emulator

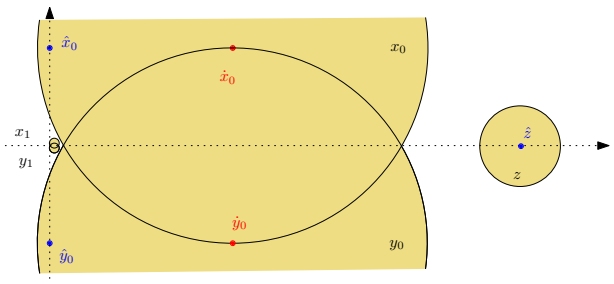

Figure 2: At most two disks overlapping with complexity $\Omega(n^2)$.

We first observe that the average numbers of triangles visited during the walk to locate a point from its hint (our method) or its predecessor in spatial sort (CGAL method) are both a small constant independent of the size of the point set. The number of visited triangles is around 2.8 for the hint and about 3.7 for spatial sort. Thus our theoretical linear complexity is confirmed by the experiments.

Unfortunately, a similar behavior is not observed for the running time. Our interpretation is that the random insertion order is demanding more and more to the cache memory management when the input size increases. Since the spatial sort order insert the new point near the previous one, relevant triangles are already loaded in the cache memory and it reaches a better running time, even if the length of the walk is longer than for our method. We combine the advantages of both methods by using the spatial sort order to preprocess the imprecise points and to process the instance with our method. The results are satisfactory and our method is clearly faster than the direct computation.

4 Beyond disjoint unit disks in the plane

The algorithm does not need the disks to be disjoint nor have unit radius, these hypotheses are only useful for proof of complexity. In fact if we allows unit disks overlapping at most k times the Delaunay triangulation of an instance is computed in $O(kn)$ time.

If the unit radius hypothesis is removed, the proof of complexity falls. Indeed, it is possible to design an example of n disks overlapping at most twice such that the algorithm takes quadratic time (see Figure 2). In that example the hint for x_i is y_i with probability $\frac{1}{2}$ (and vice versa) while $\hat{S}$ can be arranged close to the y axis so that there is i points in between x_i and y_i.

The analysis extends to higher dimensions under additional hypotheses on the data, that are usual for random incremental construction. We get: if S is such that for a random sample R of size r the expected sizes of $DT_{\dot{R}}$ and $DT_{\hat{R}}$ are both $O(r)$, then S can be preprocessed in $O(n \log n)$ time such that the Delaunay triangulation of an instance is computed $O(n)$ time. Our experiments in 3D, show a running time a bit smaller than spatial sort. The smallness of the gain is mainly due to the fact that in 3D the relative weight of point location compared to the modification of the data structure is smaller than in 2D.

Acknowledgments. Author thanks Sylvain Lazard for fruitful discussions.

References

[1] Kevin Buchin. Constructing Delaunay triangulations along space-filling curves. In *Proc. 17th European Symposium on Algorithms*, volume 5757 of *Lecture Notes Comput. Sci.*, pages 119–130. Springer-Verlag, 2009. http://www.springerlink.com/index/m17216w072m54438.pdf.

[2] Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 2011. http://www.cs.princeton.edu/~wmulzer/pubs/idt_algorithmica.pdf.

[3] Bernard Chazelle, Olivier Devillers, Ferran Hurtado, Mercè Mora, Vera Sacristán, and Monique Teillaud. Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34:39–46, 2002. http://hal.inria.fr/inria-00090664.

[4] F. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28:471–486, 1998. http://portal.acm.org/citation.cfm?id=299868.299877.

[5] Christophe Delage. Spatial sorting. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.8 edition, 2011. http://www.cgal.org/Manual/3.8/doc_html/cgal_manual/packages.html#Pkg:SpatialSorting.

[6] Olivier Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002. http://hal.inria.fr/inria-00166711.

[7] Olivier Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. Research Report 7299, INRIA, 2010. http://hal.inria.fr/inria-00485915.

[8] Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom. Theory Appl.*, 43:234–242, 2009. http://linkinghub.elsevier.com/retrieve/pii/S0925772109000583.

[9] J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator. http://www.cs.cmu.edu/~quake/triangle.html.

[10] Mariette Yvinec. 2D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.8 edition, 2011. http://www.cgal.org/Manual/3.8/doc_html/cgal_manual/packages.html#Pkg:Triangulation2.