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†

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. Our algorithm has the same asymptotic complexity as previous ones for this problem, but our algorithm is much simpler and it runs faster in practice than a direct computation of the Delaunay triangulation.

1 Introduction

A popular way to model geometric uncertainties in computational geometry, is to replace an input point by a region, e.g. a disk containing the point. We call such a region an *imprecise point*. An early work in that direction is ϵ-geometry introduced by Guibas et al. [18]. Given a set of imprecise points, we call an *instance* a set of points, each taken inside an imprecise point. Then, given a classical problem for a set of points, say the Delaunay triangulation computation, imprecise computational geometry may ask different questions, or different versions of the problem:

Strong problem. What are the pairs of points that define a Delaunay edge for all possible instances?

Weak problem. What are the pairs of points that define a Delaunay edge for at least one instance?

Instance problem. Can we preprocess the imprecise points and construct a data structure allowing to compute the Delaunay triangulation on a particular instance in a fast way?

The strong and weak approaches have been used, e.g. for convex hull computations [19, 22], but a drawback of these approaches is that they often involve intricate predicates. For example, if the original problem needs a simple predicate such as an orientation test on three points, the imprecise approach will test the position of a disk with respect to the tangent to two other disks.

In this paper, we will address the instance problem for the Delaunay triangulation computation, where the imprecise points are disks in the plane.

*This work is partly supported by ANR grant Triangles (ANR-07-BLAN-0319).

†INRIA Sophia Antipolis - Méditerranée, France.
http://www.inria.fr/sophia/members/Olivier.Devillers/

Previous work

Imprecise points Löffler and Snoeyink [21] 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 [7], which makes the result mainly theoretical. Buchin et al. [4] proposed a simpler solution, which uses the split of a Delaunay triangulation in linear time [6]. 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. Note also that recent algorithms computing a Delaunay triangulation in $o(n \log n)$ time in the word-RAM model [5] may be exploitable in the context of imprecise points.

Delaunay construction and walking strategies Let us recall a useful classical incremental algorithm to compute the Delaunay triangulation [20, 16]. When a new point is inserted in a triangulation, the triangle containing it is located by a "straight walk in the triangulation" from some starting point, visiting all triangles crossed by the line segment between the starting point and the new point [13]. Then the triangulation is modified by removing non-Delaunay triangles and filling the resulting hole with new triangles incident to the new point.

If the points are inserted in a random order, standard backward analysis [24] uses the fact that the last point is a random point, thus its expected degree is less than 6 and the expected cost of modifying the triangulation is constant. Thus the main part of the cost is proportional to the length of the walk, which is directly related to the choice of the starting point. The choice of good starting points has been addressed by many papers about randomized incremental constructions [2, 3, 9, 11, 12, 14, 15].

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 in randomized expected $O(n)$ time. Compared to previous algorithms [21, 4] the theoretical asymptotic complexity is not improved, but the proposed algorithm is much simpler. It is so simple that its description fits in a dozen of lines of text.

Moreover, the algorithm is quite efficient in practice and uses only the classical predicates for Delaunay triangulation. Our experiments show that we can process an instance much faster than with the Delaunay hierarchy [12, 26] and faster than with spatial sorting [11, 3].

The algorithm works for any set of disks (overlapping, different radii) and generalizes

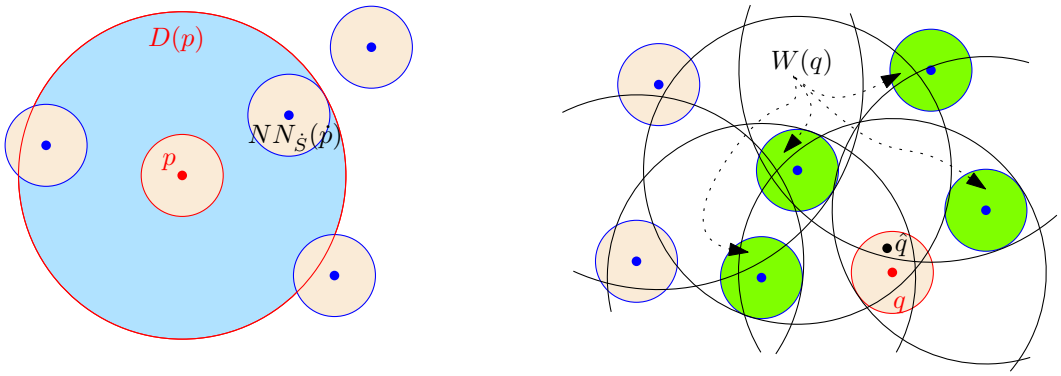

Figure 1: Left: Definition of $D(p)$ (blue disk). Right: Definition of $W(q)$ (green disks).

to balls in higher dimensions, but to yield to a good complexity the analysis requires that the imprecise 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 needs a quadratic time. The analysis for disjoint disks of different radii remains to be done.

2 Notation

- $|W|$ denotes the size of a set W.
- $||xy||$ denotes the distance between two points x and y.
- 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. Given a set of disks $S = \{p_1, p_2 \dots p_n\}$ (imprecise points), $\dot{S} = \{\dot{p}_1, \dot{p}_2 \dots \dot{p}_n\}$ and $\hat{S} = \{\hat{p}_1, \hat{p}_2 \dots \hat{p}_n\}$. The set of the first k disks is denoted $S_k = \{p_1, p_2 \dots p_k\}$, and we define $\dot{S}_k$ and $\hat{S}_k$ analogously.
- 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).

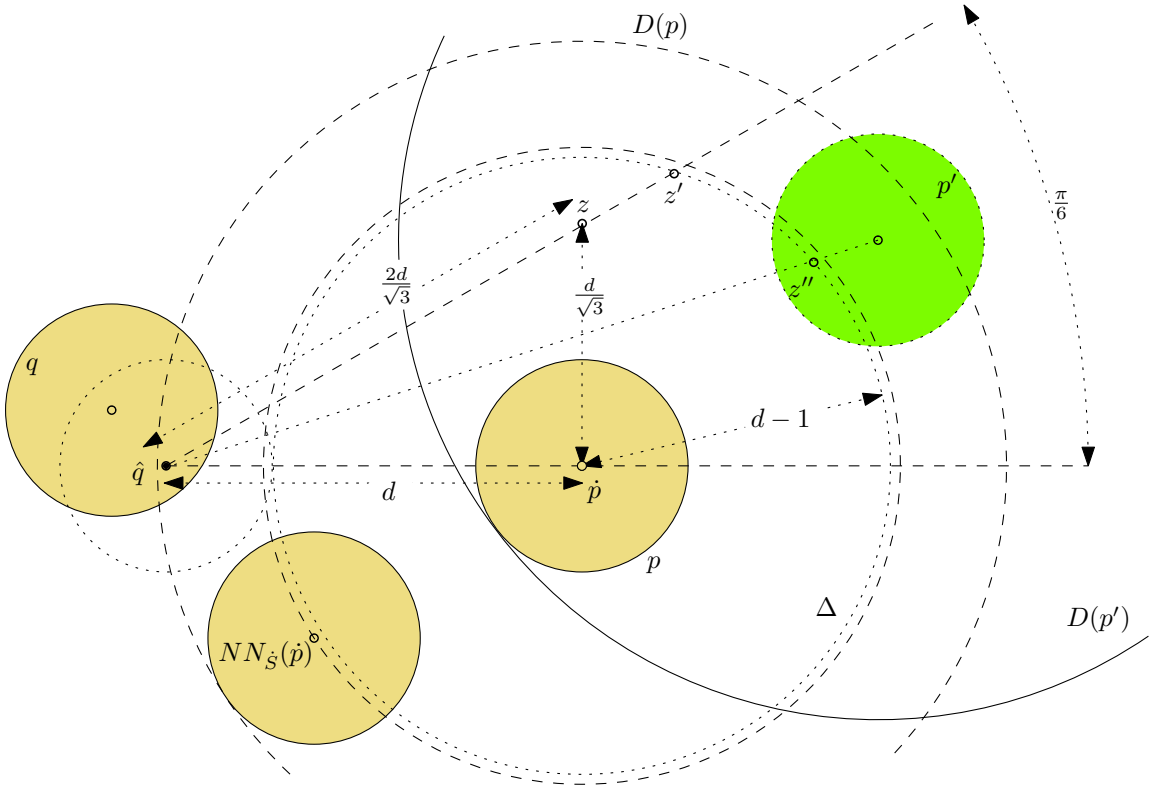

Figure 2: $D(p')$ cannot intersect q (proof of Lemma 1).

3 Algorithm

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 randomized incremental construction, it can be done in $O(n \log n)$ expected time [12] (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 $\hat{p}_{h(k)}$.

4 Complexity for unit disks

In this section, we assume that S is a set of disjoint unit disks. The proof of the linearity of the complexity starts by a lemma that bounds the size of $W(q) = \{p \in S; \hat{q} \in D(p)\}$.

Lemma 1. *$|W(q)|$ is less than or equal to 22.*

Proof. Consider $q \in S$ and a disk p such that $\hat{q} \in D(p)$. We construct the wedge of apex $\hat{q}$ limited by the ray $\hat{q}\dot{p}$ (horizontal in Figure 2) and having an apex angle of $\frac{\pi}{6}$, and let $d = ||\hat{q}\dot{p}||$. Suppose $p' \in S$ is centered in the wedge and verifies $d < ||\hat{q}\dot{p}'||$. We will prove below that if $d > 2.37$ then $D(p')$ cannot contain $\hat{q}$. Thus, in a wedge of angle $\frac{\pi}{6}$ around $\hat{q}$, among the points at distance bigger than 2.37, only the closest can belong to $W(q)$.

The size of $W(q)$ is then bounded by 12 (one point per wedge) plus the maximum number of disjoint disks at distance less than 2.37, that is $\frac{3.37^2\pi}{\pi} - 1 < 10.36$ by a simple area argument: disks with center at distance less than 2.37 are included in a disk of radius 3.37 centered at $\hat{q}$; q is counted in the area argument but does not belong to $W(q)$.

It remains to prove that $d > 2.37 \implies \hat{q} \notin D(p')$, which is done by elementary (boring) geometrical computations. Without loss of generality, $\hat{q}\dot{p}$ is assumed horizontal. We introduce (see Figure 2):
Δ : the disk of center $\dot{p}$ and radius $d-1$,
z : the vertical projection of $\dot{p}$ on the upper wedge boundary,
z' : the right most intersection of Δ and the upper wedge boundary, and
z'' : the intersection of line segment $\hat{q}\dot{p}'$ with Δ.

We make the following two remarks:
Remark 1 : Δ cannot contain any point of $\dot{S}$ (otherwise such a point would be the nearest neighbor of p and $D(p)$ would be too small to contain $\hat{q}$), and thus p' is outside Δ and z'' is well defined.
Remark 2 : Since $d > 2.37 > \frac{\sqrt{3}}{\sqrt{3}-1}$, we have $||\dot{p}z|| = \frac{d}{\sqrt{3}} < d-1 = ||\dot{p}z'||$ that is z' is to the right of z.

Then we have that the radius of $D(p')$ is

$$
\begin{aligned}
||\dot{p}'NN_{\dot{S}}(\dot{p}')|| + 1 &\leq ||\dot{p}'\dot{p}|| + 1 && \text{p cannot be closer than nearest neighbor}\\
&< ||\dot{p}'z''|| + ||z''\dot{p}|| + 1 && \text{triangular inequality}\\
&< ||\dot{p}'z''|| + (d-1) + 1 && \text{radius of } \Delta\\
&< ||\dot{p}'z''|| + \frac{2}{\sqrt{3}}d\\
&< ||\dot{p}'z''|| + ||z\hat{q}|| && \text{definition of } z\\
&< ||\dot{p}'z''|| + ||z'\hat{q}|| && \text{Remark 2: } z' \text{ to the right of } z\\
&< ||\dot{p}'z''|| + ||z''\hat{q}|| && \text{circle of center } \hat{q} \text{ through } z' \text{ cross } \Delta \text{ in } z'\\
&< ||\dot{p}'\hat{q}|| && \text{points are collinear}
\end{aligned}
$$

and thus $\hat{q} \notin D(p')$ or equivalently $p' \notin W(q)$. □

Lemma 2. *If edge $\hat{q}\hat{q}'$ of $DT_{\hat{S}}$ intersects line segment $\hat{x}\hat{p}$, with $\dot{x} = NN_{\dot{S}}(\dot{p})$, then either $\hat{q}$ or $\hat{q}'$ belongs to $D(p)$.*

Proof. By definition $D(p)$ encloses p and x and thus contains $\hat{p}$ and $\hat{x}$, and we can construct a circle $\Gamma \subset D(p)$ passing through $\hat{p}$ and $\hat{x}$. Since $\hat{q}\hat{q}' \in DT_{\hat{S}}$ we have $\hat{q}\hat{q}' \in DT_{\{\hat{q},\hat{q}',\hat{x},\hat{p}\}}$, thus $\hat{p}\hat{x}$, the other diagonal of the convex quadrilateral $\hat{q}, \hat{x}, \hat{q}'\hat{p}$, is not an edge of $DT_{\{\hat{q}\hat{q}'\hat{x}\hat{p}\}}$ or in other words a circle trough $\hat{p}$ and $\hat{x}$ cannot be empty of points of $\{\hat{q}, \hat{q}', \hat{x}, \hat{p}\}$. $\Gamma \subset D(p)$ is such a circle, thus it contains $\hat{q}$ or $\hat{q}'$. □

Theorem 3. *The expected cost of constructing $DT_{\hat{S}}$ is linear.*

Proof. 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 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 of visiting the triangles crossed by line segment $\hat{x}\hat{p}$, and
- the cost of modifying the triangulation to update $DT_{\hat{S}\setminus\{\hat{p}\}}$ into $DT_{\hat{S}}$.

The cost of turning around $\hat{x}$ is $d^{\circ}_{DT_{\hat{S}\setminus\{\hat{p}\}}}(\hat{x}) \leq d^{\circ}_{DT_{\hat{S}}}(\hat{x}) + d^{\circ}_{DT_{\hat{S}}}(\hat{p})$.

The cost of updating the triangulation is $d^{\circ}_{DT_{\hat{S}}}(\hat{p})$.

The cost of the walk is the number of edges of $DT_{\hat{S}\setminus\{\hat{p}\}}$ crossed by $\hat{x}\hat{p}$ and we distinguish between the edges that belong to $DT_{\hat{S}}$ and those that disappear in $DT_{\hat{S}}$. We bound the number of crossed edges that disappear by the total number of edges of $DT_{\hat{S}\setminus\{\hat{p}\}}$ that disappear in $DT_{\hat{S}}$, that is $d^{\circ}_{DT_{\hat{S}}}(\hat{p}) - 3$. By Lemma 2 , the edges of $DT_{\hat{S}}$ crossed by $\hat{x}\hat{p}$ have a vertex in $D(p)$. Altogether the cost of the insertion of p is bounded by

$$d^{\circ}_{DT_{\hat{S}}}(\hat{x}) + 3 \times d^{\circ}_{DT_{\hat{S}}}(\hat{p}) + \sum_{\hat{q}\in D(p)} d^{\circ}_{DT_{\hat{S}}}(\hat{q})$$

Then the three following claims end the proof of the theorem. Notice that p is the last imprecise point and, since the points are indexed in a random order, it is also a random point in S. [.] is the indicator function (value is 1 if the event is true, 0 otherwise).

Claim *The expected degree of $\hat{p}$ in $DT_{\hat{S}}$ is less than or equal to 6.*

This is just the degree of a random point in $DT_{\hat{S}}$. ◊

Claim *The expected degree of $\hat{x}$ in $DT_{\hat{S}}$ is less than or equal to 36.*

Using the well known fact that the degree of anyvertex of the nearest neighbor graph of a point set is bounded by 6,

$$\begin{aligned}
E\left(d^\circ_{DT_{\hat{S}}}(\hat{x})\right) &= \frac{1}{n}\sum_{\substack{p\in S\\ \dot{x}=NN_{\dot{S}}(\dot{p})}} d^\circ_{DT_{\hat{S}}}(\hat{x}) \\
&= \frac{1}{n}\sum_{p\in S}\sum_{x\in S}[\dot{x}=NN_{\dot{S}}(\dot{p})]d^\circ_{DT_{\hat{S}}}(\hat{x}) \\
&= \frac{1}{n}\sum_{x\in S} d^\circ_{DT_{\hat{S}}}(\hat{x})\sum_{p\in S}[\dot{x}=NN_{\dot{S}}(\dot{p})] \\
&= \frac{1}{n}\sum_{x\in S} d^\circ_{DT_{\hat{S}}}(\hat{x}) d^\circ_{NN_{\dot{S}}}(\dot{x}) \\
&\leq \frac{6}{n}\sum_{x\in S} d^\circ_{DT_{\hat{S}}}(\hat{x}) \leq \frac{6}{n}\,6n = 36.
\end{aligned}$$

◊

Claim *The expected value of $\sum_{\hat{q}\in D(p)} d^\circ_{DT_{\hat{S}}}(\hat{q})$ is less than or equal to 132.*

$$\begin{aligned}
E\left(\sum_{\hat{q}\in D(p)} d^\circ_{DT_{\hat{S}}}(\hat{q})\right) &= \frac{1}{n}\sum_{p\in S}\sum_{\hat{q}\in D(p)} d^\circ_{DT_{\hat{S}}}(\hat{q}) \\
&= \frac{1}{n}\sum_{p\in S}\sum_{q\in S}[\hat{q}\in D(p)]d^\circ_{DT_{\hat{S}}}(\hat{q}) \\
&= \frac{1}{n}\sum_{q\in S} d^\circ_{DT_{\hat{S}}}(\hat{q})\sum_{p\in S}[p\in W(q)] \\
&= \frac{1}{n}\sum_{q\in S} |W(q)| d^\circ_{DT_{\hat{S}}}(\hat{q}) \qquad \text{then by Lemma 1} \\
&\leq \frac{22}{n}\sum_{q\in S} d^\circ_{DT_{\hat{S}}}(\hat{q}) \leq \frac{22}{n}\,6n \leq 132.
\end{aligned}$$

◊

This ends the proof of Theorem 3. □

5 Experiments

Algorithms proposed in previous works [4, 21] 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 preprocessing the imprecise points.

We compare our approach with the best implementations available in CGAL [26] and with Shewchuk's code: Triangle [25].

- **hierarchy** (dynamic method): the Delaunay hierarchy allows one to insert points dynamically [12].
- **spatial sort** (static method): the points are ordered along a space filling curve and then inserted in the *spatial sort* order [11], 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).
- **Shewchuk**: Shewchuk's code uses a divide and conquer method.

Since our approach is also implemented using CGAL, the comparison with spatial sort really measures the influence of the starting point of the walk and not the implementation of the walk itself.

Our method is referenced in the tables as **hint random order** or **hint spatial sort** for a variant introduced below.

Point sets

We experiment on several kinds of point sets, mostly synthetic, except from the 3D scanned models. These sets are not intended to represent actual applications but to be an interesting and easy to reproduce benchmark.

- **Random disks.**

  These point sets stay in the strict hypotheses of the theoretical analysis, that is: 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 2. 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; we generate 100 point sets of sizes between 10^3 and 10^6 and 10 point sets of size 10^7 and 10^8 and one instance for each point set.

- **Brownian motion.**

  Updating the Delaunay triangulation of moving points is a difficult task [8, 17] that arises in mesh optimization [10] or fluid dynamics [1].

  To simulate this kind of data, we generate a random point set of n points in a $4\sqrt{n} \times 4\sqrt{n}$ square (unit disks centered at these points may not be disjoint). Then at each time step, each point is moved randomly in a unit disk around its previous location. Point sets with size ranging between 10^3 and 10^8 points have been generated.

Figure 3: Scanned models (from Aim@Shape repository).

- **Random balls.**

  The "Random disks" point sets have also been generated in dimension three with similar constraints.

- **3D noisy data.**

  We start from a point set originating from a laser scan of a 3D object. Then, the centers of our n imprecise points is a random sample of the scan, and an instance is a noisy version of these points. To adapt to the model size, the noise intensity (the radius of the ball) is fixed at the average distance between an imprecise point and its hint. The original data are depicted on Figure 3 and are courtesy of Aim@Shape (Neptune, Galaad and Olivier's hand on http://www.aimatshape.net/). Files of smaller size are obtained by taking a random sample of the original data set.

Platform

Experiments have been done on the following platform:

- CGAL 3.8 compiled with `gcc 4.3.2` in release mode,
- Shewchuk's code is compiled with `-O3` (faster than `-O2` on this platform),
- 3.00 GHz processor, 32 GB RAM, 6 MB L2 cache, and 4KB pagesize,
- Linux-FC10,
- timings obtained with the `CGAL::Timer`.

Source code is joined to this paper on the journal web site.

Results

In the following tables, we report the time for computing the Delaunay triangulation and the number of triangles visited during the straight walks used in point locations. These numbers are averaged on several trials, and the standard deviation is given. These numbers are divided by the number of points.

We first observe that the average numbers of triangles visited during the walk to locate a point from its hint or its predecessor in spatial sort (CGAL static 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, proved in Theorem 3, is confirmed by the experiments.

Unfortunately, a similar behavior is not observed for the running time. While the small increase in the time for computing the Delaunay triangulation in spatial sort order is well explained by the time of sorting itself, the running time of our method increases in a super linear way contradicting our expectation.

2D random imprecise points	running time ⊕ standard deviation per point (μs) ♯ visited triangles ⊕ standard deviation per point					
n	10^3	10^4	10^5	10^6	10^7	10^8
♯ trials	100	100	100	100	10	10
hint	0.9 ⊕ 0.3	0.88 ⊕ 0.1	1.2 ⊕ 0.03	2.9 ⊕ 0.01	3.8 ⊕ 0.005	5.4 ⊕ 0.009
random order	2.83 ⊕ 0.05	2.80 ⊕ 0.01	2.77 ⊕ 0.005	2.75 ⊕ 0.001	2.75 ⊕ 0.0005	2.74 ⊕ 0.0002
spatial sort	1.1 ⊕ 0.5	0.85 ⊕ 0.01	0.83 ⊕ 0.1	0.90 ⊕ 0.01	1.0 ⊕ 0.004	1.13 ⊕ 0.004
	3.74 ⊕ 0.07	3.63 ⊕ 0.02	3.71 ⊕ 0.007	3.67 ⊕ 0.003	3.55 ⊕ 0.0006	3.71 ⊕ 0.0002
Delaunay	1.8 ⊕ 0.4	1.6 ⊕ 0.2	2.8 ⊕ 0.04	5.78 ⊕ 0.02	9.0 ⊕ 0.02	13 ⊕ 0.04
hierarchy	24 ⊕ 0.6	28 ⊕ 0.6	29 ⊕ 0.6	38 ⊕ 0.5	45 ⊕ 0.6	47 ⊕ 0.6
Shewchuk	0.96 ⊕ 0.2	1.12 ⊕ 0.1	1.05 ⊕ 0.006	1.61 ⊕ 0.02	2.4 ⊕ 0.03	
hint	1.0 ⊕ 0.5	0.79 ⊕ 0.1	0.59 ⊕ 0.02	0.61 ⊕ 0.009	0.61 ⊕ 0.004	0.62 ⊕ 0.002
spatial sort	2.82 ⊕ 0.05	2.80 ⊕ 0.2	2.77 ⊕ 0.004	2.76 ⊕ 0.002	2.75 ⊕ 0.0004	2.75 ⊕ 0.0002
preprocessing (s)	0.002 ⊕ 0.0003	0.014 ⊕ 0.0004	0.17 ⊕ 0.002	2.0 ⊕ 0.01	23 ⊕ 0.06	270 ⊕ 0.3
♯ instances to amortize			15	7	6	6

Our interpretation is that the random insertion order is demanding more and more cache memory management as the input size increases. Since the spatial sort order inserts 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. This variant is referenced as **hint spatial sort** in the tables. The results are satisfactory and this variant is significantly faster than the direct computation. For large point sets, processing 6 or 7 instances are enough to amortize the cost of the preprocessing.

The involvement of the cache memory management in the discrepancy between the number of visited triangles and the actual running time is confirmed by the number of cache

misses obtained by running our code under the emulator[1] `valgrind --tool=cachegrind`. Cache misses are reported in the following table. Starting at 50000 points, the number of cache misses increases much more when the hint in random order is used than when the spatial sort order is used (with or without hint). The number of cache misses is given for the whole program, and thus some cache misses occurring during the initialization are amortized on the number of points, which explain the high values for small input.

2D random	number of cache misses per point						
n	1000	10 000	25 000	50 000	100 000	1 000 000	10 000 000
hint	14	3.5	2.8	4.3	9.7	27	32
spatial sort	14	3.6	2.9	3.0	3.1	5.3	7.7
hint spatial sort	14	3.5	2.8	3.1	3.3	3.8	3.8
Shewchuk	5.3	2.3	2.1	2.2	2.8	20	39

Above results are confirmed by the Brownian motion experiment. As a basis for comparison, the Delaunay triangulation is computed using spatial sort ordering at each time step. For our method, we order the points using spatial sort on the initial position. Then at each time step the points are inserted in the same order, locating from the hint. For each inserted point, its new nearest neighbor is computed and stored as the hint for the next time step. As previously observed, our method is really faster for large point sets. The total time in the table below includes the motion generation and the initial preprocessing for the hint method.

2D Brownian motion	running time (μs) per point per time step ♯ visited triangles per point per time step total time (s)					
n ♯ steps	10^3 100	10^4 100	10^5 100	10^6 100	10^7 100	10^8 10
spatial sort	0.78 ⊕ 0.4 3.81 ⊕ 0.06 0.088	0.78 ⊕ 0.5 3.68 ⊕ 0.02 0.85	0.88 ⊕ 0.02 3.77 ⊕ 0.006 9.4	0.96 ⊕ 0.02 3.72 ⊕ 0.002 102	1.12 ⊕ 0.008 3.62 ⊕ 0.001 1134	1.20 ⊕ 0.02 3.78 ⊕ 0.0002 1223
hint spatial sort	0.73 ⊕ 0.5 2.77 ⊕ 0.04 0.083	0.69 ⊕ 0.3 2.77 ⊕ 0.01 0.76	0.79 ⊕ 0.01 2.77 ⊕ 0.006 8.6	0.81 ⊕ 0.02 2.77 ⊕ 0.003 86	0.83 ⊕ 0.005 2.77 ⊕ 0.002 861	0.82 ⊕ 0.02 2.77 ⊕ 0.0007 894

For points in 3D, CGAL offers (as in 2D) spatial sorting, point location by walk, and Delaunay hierarchy [23]; thus we run similar experiments in 3D and observe a similar advantage for our point location from the hint. But the smaller weight of the point location in the whole insertion process in 3D, makes this advantage less important. The final improvement on the running time is around 5%.

[1] This emulator ran on a similar machine with only 4MB cache size.

3D random imprecise points	running time (μs) per point ♯ visited triangles per point				
n ♯ trials	10^3 100	10^4 100	10^5 100	10^6 100	10^7 10
hint	9.2 ⊕ 1.6 5.2 ⊕ 0.08	7.8 ⊕ 0.2 5.3 ⊕ 0.03	14.2 ⊕ 0.07 5.3 ⊕ 0.008	19 ⊕ 0.04 5.2 ⊕ 0.003	23 ⊕ 0.03 5.2 ⊕ 0.0007
spatial sort	9.0 ⊕ 1.6 6.3 ⊕ 0.1	7.6 ⊕ 0.2 6.6 ⊕ 0.04	8.0 ⊕ 0.06 6.6 ⊕ 0.01	8.2 ⊕ 0.04 6.6 ⊕ 0.004	8.4 ⊕ 0.04 6.6 ⊕ 0.001
Delaunay hierarchy	11 ⊕ 1.2 21 ⊕ 0.4	9.7 ⊕ 0.2 29 ⊕ 0.4	18 ⊕ 0.07 34 ⊕ 0.5	25 ⊕ 0.06 42 ⊕ 0.4	33 ⊕ 0.06 50 ⊕ 0.3
hint spatial sort	9.5 ⊕ 0.9 4.4 ⊕ 0.09	7.5 ⊕ 0.2 4.6 ⊕ 0.03	7.8 ⊕ 0.07 4.5 ⊕ 0.009	7.9 ⊕ 0.04 4.5 ⊕ 0.003	8.0 ⊕ 0.05 4.4 ⊕ 0.0009

The results on more realistic data are similar and we still observe an improvement of a few % on the running time.

3D noisy sample of scanned models	running time (μs) per point ♯ visited triangles per point			
n	10^3	10^4	10^5	full size
Hand				53 Kpoints
spatial sort	8 7.0	8.4 8.1		8.2 8.1
hint spatial sort	6 5.3	7.3 5.8		7.6 6.1
Galaad				1.45 Mpoints
spatial sort	8 7.5	7.9 8.1	7.9 8.1	8.2 9.0
hint spatial sort	6 5.7	7.2 5.8	7.6 5.9	7.7 6.1
Neptune				2 Mpoints
spatial sort	7 7.0	8.2 8.0	8.6 8.6	8.9 9.5
hint spatial sort	7 5.7	7.5 6.0	7.7 6.1	7.5 6.4

6 Beyond disjoint unit disks in the plane

6.1 Overlapping unit disks

The algorithm does not need the disks to be disjoint and have unit radius, these hypotheses are only useful for Lemma 1. The wedge argument used in the proof of Lemma 1 does not use the fact that the disks are non overlapping, thus, if the disks overlap at most k times, by a straightforward modification, Lemma 1 generalizes in $|W(q)| < 11.36\,k + 12 = O(k)$. Thus we get that n unit disks that overlap at most k times can be preprocessed in $O(n \log n)$ time such that the Delaunay triangulation of an instance is computed in $O(kn)$ time.

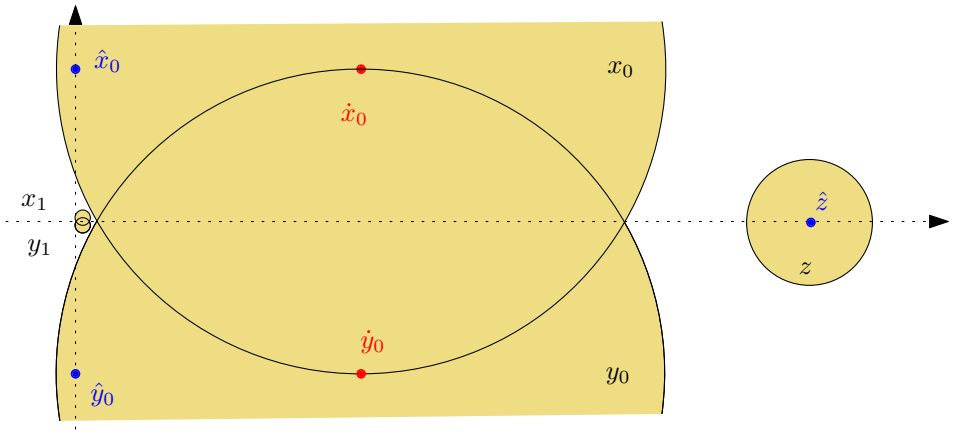

Figure 4: For disks of different radii with at most two overlapping, the complexity is $\Omega(n^2)$.

6.2 Non-unit disks

Unfortunately, if the disks have different radii, Lemma 1 does not generalize. We have a counter example that, if we allow at most two disks to overlap, the complexity of our algorithm becomes quadratic. Consider the following example of $2n+1$ disks (see Figure 4): x_i is centered at $10^{-i} \cdot (2,1)$ and has radius $10^{-i}2.1$ for $0 \leq i < n$ and $\hat{x}_i$ is placed at $10^{-i}(2,\epsilon^i)$; y_i is symmetric to x_i with respect to horizontal axis and z is on the x-axis with $\hat{z} = \dot{z} = (0,3)$. Then the points are inserted in a random order (unrelated to the order of index i), to construct $DT_{\dot{S}}$. Now when an instance is processed, after the insertion of $\hat{z}$, $DT_{\dot{S}}$ links $\hat{z}$ to all other already inserted points (all $\hat{x}_i$ and $\hat{y}_j$ are almost vertically collinear). Then the location of a new point $\dot{x}_i$ starts at its nearest neighbor which is $\dot{y}_i$ if $\dot{y}_i$ is already inserted, that is with probability $\frac{1}{2}$. In such a case, the walk crosses all edges $\hat{z}\hat{x}_j$, $\hat{z}\hat{y}_k$ for $j,k \geq i$ if these points are already inserted. Thus, with constant probability, the insertion of a point takes linear time, and thus the total expected time for constructing the Delaunay triangulation is quadratic. If the disks have different sizes and do not overlap, the complexity of our algorithm remains open.

6.3 Higher dimension

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.

Acknowledgments Author thanks Sylvain Lazard for fruitful discussions, anonymous referees for their helpful comments, and Ramsay Dyer for his help in preparing the final version of this paper.

References

[1] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Transactions on Graphics*, 26, 2007. Proceedings of ACM SIGGRAPH, http://dx.doi.org/10.1145/1276377.1276437.

[2] Nina Amenta, Sunghee Choi, and Günter Rote. Incremental constructions con BRIO. In *Proc. 19th Annu. Sympos. Comput. Geom.*, pages 211–219, 2003. http://page.inf.fu-berlin.de/~rote/Papers/pdf/Incremental+constructions+con+BRIO.pdf.

[3] 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.

[4] Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica, online first*, 2011. http://www.springerlink.com/content/68033136048028h9/fulltext.pdf.

[5] Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in o(sort(n)) time and more. In *Proc. 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 139–148, 2009. http://www.cs.princeton.edu/~wmulzer/pubs/wramdtFOCS.pdf.

[6] 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.

[7] Francis Chin and Cao An 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.

[8] Pedro Machado Manhães de Castro. *Practical Ways to Accelerate Delaunay Triangulations*. Thèse de doctorat en sciences, Université de Nice Sophia Antipolis, France, 2010. http://tel.archives-ouvertes.fr/tel-00531765/fr/.

[9] Pedro Machado Manhães de Castro and Olivier Devillers. Simple and efficient distribution-sensitive point location, in triangulations. In *Workshop on Algorithm Engineering and Experiments*, 2011. http://www.siam.org/proceedings/alenex/2011/alenex11.php.

[10] Pedro Machado Manhães de Castro, Jane Tournois, Pierre Alliez, and Olivier Devillers. Filtering relocations on a Delaunay triangulation. *Computer Graphics Forum*, 28:1465–1474, 2009. Special issue 6th Annu. Sympos. Geometry Processing, http://hal.inria.fr/inria-00413344.

[11] 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.

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

[13] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *Internat. J. Found. Comput. Sci.*, 13:181–199, 2002. http://hal.inria.fr/inria-00102194/.

[14] Luc Devroye, Christophe Lemaire, and Jean-Michel Moreau. Expected time analysis for Delaunay point location. *Comput. Geom. Theory Appl.*, 29:61–89, 2004. http://cg.scs.carleton.ca/~luc/CGTA63-final.pdf.

[15] Luc Devroye, Ernst Peter Mücke, and Binhai Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998. http://cg.scs.carleton.ca/~luc/devroye_mucke_zhu_1998_a_note_on_point_location_in_delaunay_triangulations_of_random_points.pdf.

[16] P. J. Green and R. R. Sibson. Computing Dirichlet tessellations in the plane. *Comput. J.*, 21:168–173, 1978. http://comjnl.oxfordjournals.org/content/21/2/168.abstract.

[17] Leonidas Guibas and Daniel Russel. An empirical comparison of techniques for updating Delaunay triangulations. In *Proc. 20nd Annu. Sympos. Comput. Geom.*, pages 170–179, 2004. http://dx.doi.org/10.1145/997817.997846.

[18] Leonidas J. Guibas, David Salesin, and Jorge Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. Sympos. Comput. Geom.*, pages 208–217, 1989. http://portal.acm.org/citation.cfm?id=73833.73857&type=series.

[19] Leonidas J. Guibas, David Salesin, and Jorge Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993. http://www.springerlink.com/index/T7371085XLW37RP1.pdf.

[20] Charles L. Lawson. C^1 surface interpolation for scattered data on a sphere. *Rocky Mountain J. Math.*, 14(1):177–202, 1984. http://rmmc.asu.edu/TO%20DOUGLAS/RMJ/vol14/vol14-1/law.pdf.

[21] 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.

[22] Takayuki Nagai, Seigo Yasutome, and Nobuki Tokura. Convex hull problem with imprecise input. In *Japanese Conference on Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 207–219. Springer-Verlag, 2004. http://www.springerlink.com/index/509GGJTDXL14AEBN.pdf.

[23] Sylvain Pion and Monique Teillaud. 3D 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:Triangulation3.

[24] Raimund Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993. http://ftp.icsi.berkeley.edu/ftp/pub/techreports/1992/tr-92-014.ps.gz.

[25] Jonathan Richard Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in*

Computer Science, pages 203–222. Springer-Verlag, 1996.
http://www.cs.cmu.edu/~quake-papers/triangle.ps.

[26] 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.