Practical Distribution-Sensitive Point Location in Triangulations

Pedro Machado Manhães de Castro, Olivier Devillers

Practical Distribution-Sensitive Point Location in Triangulations*

Pedro Machado Manhães de Castro† Olivier Devillers‡

Computer Aided Geometric Design 30 (2013) 431–450

Abstract

We design, analyze, implement, and evaluate a distribution-sensitive point location algorithm based on the classical Jump & Walk, called Keep, Jump, & Walk. For a batch of query points, the main idea is to use previous queries to improve the current one. In practice, Keep, Jump, & Walk is actually a very competitive method to locate points in a triangulation. We also study some constant-memory distribution-sensitive point location algorithms, which work well in practice with the classical space-filling heuristic for fast point location. Regarding point location in a Delaunay triangulation, we show how the Delaunay hierarchy can be used to answer, under some hypotheses, a query q with a $O(\log \#(pq))$ randomized expected complexity, where p is a previously located query and $\#(s)$ indicates the number of simplices crossed by the line segment s. The Delaunay hierarchy has $O(n \log n)$ time complexity and $O(n)$ memory complexity in the plane, and under certain realistic hypotheses these complexities generalize to any finite dimension. Finally, we combine the good distribution-sensitive behavior of Keep, Jump, & Walk, and the good complexity of the Delaunay hierarchy, into a novel point location algorithm called Keep, Jump, & Climb. To the best of our knowledge, Keep, Jump, & Climb is the first practical distribution-sensitive algorithm that works both in theory and in practice for Delaunay triangulations.

1 Introduction

Point location in spatial subdivision is one of the most classical problems in computational geometry [27]. Given a query-point q and a partition of the d-dimensional space in regions, the problem is to retrieve the region containing q. This paper addresses the special case where the spatial subdivision is a simplicial complex, also called *triangulation*.

In two dimensions, locating a point has been solved in optimal $O(n)$ space and $O(\log n)$ worst-case query time a quarter of a century ago by Kirkpatrick's hierarchy [36]. In three dimensions, optimal point location remains an open problem [17]. While $O(\log n)$ is the best worst-case query time one can guarantee, it turns out that it is possible to improve the query time when successive queries have some spatial coherence. For instance, spatial coherence occurs (i) when the queries follow some specific path inside a region, (ii) when a method (e.g., the Poisson surface reconstruction [34, 11]) uses point dichotomy to find the solution to some equation, or (iii) in geographic information systems, where the data base contains some huge geographic area, while the queries lie in some small region of interest. During the last twenty-five years, computer geometers borrowed from the classical one-dimensional framework [37, 30], two ways to take the spatial coherence into account in point location algorithms: (i) using the *entropy* of the query distribution [5, 32], and (ii) designing algorithms that have a self-adapting capability, i.e. algorithms that are *distribution-sensitive* [31, 18].

Entropy. Entropy-based point location assumes that a distribution on the set of queries is known. There are some well-known entropy-based point location data structures in two dimensions. Arya *et al.* [5]

*This work is partially supported by ANR Project *Triangles* and Région PACA.

†Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE), Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária (Campus Recife) 50.740-560 - Recife - Pernambuco - BRASIL. pmmc@cin.ufpe.br

‡INRIA, 2004 route des lucioles BP 93, 06902 Sophia Antipolis - FRANCE. Olivier.Devillers@inria.fr

or Iacono [32], both achieve a query time proportional to the entropy of that distribution, linear space, and $O(n \log n)$ preprocessing time. Those algorithms are asymptotically optimal [38]. However, in many applications the distribution is unknown. Moreover, the distribution of query points can deliberately change over time. Still, it is possible to have a better complexity than the worst-case optimal if queries are very close to each other.

Distribution-sensitiveness. A point location algorithm that adapts to the distribution of the query is called a *distribution-sensitive* point location algorithm. A few distribution-sensitive point location algorithms in the plane exist: Iacono and Langerman [31] and Demaine *et al.* [18]. Both achieve a query time that is logarithmic in terms of the distance between two successive queries for some special distances. However the space required is above linear, and preprocessing time is above $O(n \log n)$. More recently, a 2D point location algorithm that theoretically behaves as good as the best entropy-based algorithms (in an amortized sense) without *a priori* knowledge of query distribution has been proposed by Iacono [33] in SoCG 2011.

Walk. Despite the good theoretical query time of the point location algorithms above, alternative algorithms using simpler data structures are still used by practitioners. Amongst these algorithms, *walking* from a simplex to another using the neighborhood relationships between simplices, is a straightforward algorithm which does not need any additional data structure besides the triangulation [20]. Walking performs well in practice for Delaunay triangulations, but has a non-optimal complexity [21].

Building on walk. Building on the simplicity of the walk, both the Jump & Walk [39] and the Delaunay hierarchy [19] improve the complexity while retaining the simplicity of the data structure. The main idea of these two structures is to find a good starting point for the walk in order to reduce the number of visited simplices [39, 12, 19, 49]. In two dimensions, Jump & Walk guarantees randomized $O(\sqrt[3]{n})$ query time for random Delaunay triangulations with no other preprocessing than the computation of the Delaunay triangulation [1], whereas the Delaunay hierarchy guarantees a randomized $O(\log n)$ worst-case query time for any Delaunay triangulation, but with $O(n \log n)$ preprocessing. Furthermore, these methods based on walking extend well for any finite dimension, which is not true for the aforementioned optimal algorithms. Under some realistic hypotheses, the Delaunay hierarchy guarantees a randomized $O(\log n)$ worst-case query time even for Delaunay triangulations in the d-dimensional space [19]. Delaunay hierarchy is currently implemented as the `Fast_location` policy of the Computational Geometry Algorithms Library [11, 48, 40] (CGAL).

Contribution. In this paper, we propose several new ideas to improve point location in practice. Under some hypotheses satisfied by "real point sets", we also obtain interesting theoretical analysis.

In Section 3, we introduce the *Distribution Condition*: A region $\mathcal{C}$ of a triangulation $\mathcal{T}$ satisfies this condition if the expected cost of walking in $\mathcal{T}$ along a segment inside $\mathcal{C}$ is in the worst case proportional to the length of this segment. In Section 7.1, we provide experimental evidence that some realistic triangulations satisfy the Distribution Condition for the whole region inside their *convex hull*. And, we relate this condition to the length of the edges of some spanning trees embedded in $\mathbb{R}^d$, to obtain complexity results. Section 3.1 reviews some of the most common forms of trees embedded in the space. We recall some interesting results concerning their lengths, which are useful in the sequel.

In Section 4, we investigate constant-size-memory strategies for choosing the starting point of a walk. More precisely, we compare strategies that are dependent on previous queries and strategies that are not, mainly in the case of random queries. Random queries are, *a priori*, not favorable to distribution-sensitive strategies, since there is no coherence between them. Nevertheless, our computations prove that distribution-sensitive strategies are, either better, or not dramatically worse in this case. Thus, there is a good reason for the use of distribution-sensitive strategies, since they are competitive even in situations that are seemingly unfavorable. Section 7.2 provides experiments to confirm such behavior on realistic data.

In Section 5, we revisit Jump & Walk so as to make it distribution-sensitive. The modification is called Keep, Jump, & Walk. In a different setting, Haran and Halperin verified experimentally [29] that similar ideas in the plane gives interesting running time in practice. Here, we give theoretical guarantees that, under some conditions, the expected amortized complexity of Keep, Jump, & Walk is the same as the

[1] Actually, the Delaunay triangulation been given as input, Jump & Walk belongs to the "sub-linear geometric algorithm" category introduced by Chazelle et al. [15].

expected complexity of the classical Jump & Walk. We also provide analysis for some modified versions of Keep, Jump, & Walk.

In Section 7.4, experiments show that Keep, Jump, & Walk, has an improved performance compared to the classical Jump & Walk in 3D as well. Despite its simplicity, it is a competitive method to locate points in a triangulation.

In Section 6, we show that *climbing* the Delaunay hierarchy can be used to answer a query q in $O(\log \#(pq))$ randomized expected complexity, where p is a point with a known location and $\#(s)$ indicates the expected number of simplices crossed by the line segment s. Climbing instead of walking makes Keep, Jump, & Walk become Keep, Jump, & Climb, which appears to take the best of all methods both in theory and in practice.

Table 1 gives a succinct summary of previous works and our contributions.

	Jump & Walk [39, 22]	**Keep, Jump & Walk (Section 5)**	Delaunay hierarchy [19]	**Keep, Jump, & Climb (Section 6)**
distribution-sensitive	no	**yes**	no	**yes**
expected query complexity	$O(n^{1/(d+1)})$	$\mathbf{O(n^{1/(d+1)})}$	$O(\log n)$	$\mathbf{O(\log \#(pq))}$
time to answer 2^{20} queries	8s to 25s	**5s to 10s**	5s to 12s	**1s to 11s**

Table 1: **Overview of paper's contributions.** *New results from this paper are in boldface, the ranges of time correspond to the performance obtained for different scenarios; more details on the experiments are available in Section 7.*

2 Walking in a Triangulation

2.1 Walking Strategies

First, let us define the *visibility graph* $\mathcal{VG}(\mathcal{T}, q)$ of a triangulation $\mathcal{T}$ of n points in dimension d and a query point q. $\mathcal{VG}(\mathcal{T}, q)$ (or simply $\mathcal{VG}$ when there is no ambiguity) is a directed graph (V, E), where V is the set of d-simplices of $\mathcal{T}$, and a pair of simplices (σ_i, σ_j) belongs to the set of edges E if σ_i and σ_j are adjacent in $\mathcal{T}$ and the supporting hyperplane of their common facet *separates* the interior of σ_i from q; see Figure 1. When two simplices σ_i and σ_j are such that $(\sigma_i, \sigma_j) \in E$, we will say that σ_j is a *successor* of σ_i. Now, a *visibility walk* (also called Lawson's oriented walk) consists in repeatedly walking from a simplex σ_i to one of its successor in $\mathcal{VG}$ until the simplex containing q is found; a *walking strategy* describes how this successor is chosen.

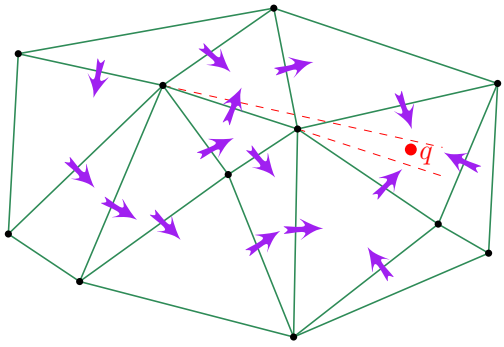


Figure 1: **Visibility graph.** *Arrows represent edges of $\mathcal{VG}$.*

The following two walking strategies will be considered: (i) the *straight walk* is a visibility walk where each visited simplex intersects the segment pq; and (ii) the *stochastic walk* is a visibility walk where the next visited simplex, from a simplex σ, is randomly chosen amongst the successors of σ in $\mathcal{VG}$.

The straight walk has a worst-case complexity linear in the number of simplices [43]. If $\mathcal{T}$ is the Delaunay triangulation of points evenly distributed in some finite convex domain and s is *not close* to the domain boundary, the expected number of simplices stabbed by a segment s is $O(\|s\| \cdot n^{1/d})$, for $d = 2$ [21, 9], for $d = 3$ [39], and believed to be true for any d [28, 10]. Current results on the complexity of the stochastic walk are weaker: It is known that the stochastic walk finishes with probability 1 [20], though it may visit an exponential number of simplices (even in $\mathbb{R}^2$). In the case of Delaunay triangulations, the complexity improves to become linear in the number of simplices. For stochastic walk on Delaunay triangulations of evenly distributed points, we have the following conjecture.

Conjecture 1. *If $\mathcal{T}$ is the Delaunay triangulation of points evenly distributed in some finite convex domain $\mathcal{C}$ and s a segment inside $\mathcal{C}$, then the number of simplices visited during the stochastic walk is in $O(1+\|s\|\cdot n^{1/d})$.*

In practice, stochastic walk answers point location queries faster than the straight walk [20] and it is the current choice of CGAL for the walking strategy [48, 40], in both dimensions 2 and 3.

In this paper, our theoretical results work for any walking procedure such that the number of simplices visited during the walk is in $O(1+\|s\|\cdot n^{1/d})$. As mentioned, this is verified for the straight walk in dimensions $d = 2, 3$, and conjectured for the straight and stochastic walk in any dimension. It has been empirically verified in [20] that the stochastic walk is faster than the straight walk, easier to implement, and more robust to numerical errors. Henceforth, we use the stochastic walk for experiments (Section 7).

2.2 Orientation Predicate and its Exact Evaluation

Given two adjacent simplices, deciding which one is a successor of the other relies on the so-called *orientation predicate*: Given $d+1$ points

$$p_0 = (x_0^0, \ldots, x_{d-1}^0), \ldots, p_d = (x_0^d, \ldots, x_{d-1}^d),$$

the orientation predicate is defined as the sign of the following determinant.

$$orient(p_0, \ldots, p_d) = sign \begin{vmatrix} x_0^0 & \ldots & x_{d-1}^0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_0^d & \ldots & x_{d-1}^d & 1 \end{vmatrix} \in \{-1, 0, +1\}. \quad (1)$$

The polynomial induced by the determinant, from which we extract the sign for a given input, is called the *predicate polynomial*.

It is commonly assumed that the representation of a simplex σ of $\mathcal{T}$ gives its vertices $p_0, \ldots, p_d$ in an order corresponding to positive orientation

$$orient(p_0, p_1, \ldots, p_d) = +1.$$

Then, an adjacent simplex σ' is the successor of σ if the supporting hyperplane of their common facet (say the facet containing $p_1, \ldots, p_d$) separates p_0 and q, which is true if $orient(q, p_1, \ldots, p_d) = -1$.

It is well known that, as for many computational geometry algorithms, an implementation of the predicate with floating point arithmetic may yield to robustness issues [35]; concerning the walk, the problem is that it may never terminate. The exact geometric computation paradigm [47] provides one interesting solution: evaluating the predicate exactly. However, since an exact evaluation of the predicate polynomial is quite expensive the trick is to have a *filtered evaluation* that is an approximate evaluation of the predicate polynomial with a certification of the exactness of its sign. If the certification is successful, the predicate is reasonably cheap. Otherwise, the expensive exact computation is called, but it does not arise often.

In the context of CGAL the approximate evaluation consists in computing the predicate polynomial with usual floating point arithmetic to produce a result δ. The certification computes a bound β on the maximal error between the inexact computation and the exact value, if $|\delta| > \beta$, then the inexact computation is certified, and we are done; otherwise, an exact computation using *multi-precision type* [1, 3, 2] is performed.

2.3 Optimizing through Structural Filtering

As mentioned, the exactness of the walk is certified by exact computations, which are classically accelerated with arithmetic filtering. To speed-up the walking procedure even more, we can use a filtering scheme called *structural filtering*, proposed by Funke et al. [24], which works at a higher level than the classical arithmetic filtering scheme. It is based on the *relaxation* of the exactness of the predicates.

More precisely, the correctness of the visibility walk relies on the exactness of the orientation predicates. The filtering mechanism accelerates the exact computation of the orientation predicate (see Section 2.2), but it remains slower than a direct evaluation with floating-point numbers without any certification of the sign at all. Structural filtering extends this classical filtering mechanism to a higher level. Instead of certifying exactness at each orientation predicate, the process just certifies that the returned simplex contains the query. Since a walk is mostly dependent on the performance of the orientation predicate, using a cheaper predicate will improve the whole performance of the walk. Therefore, a two-phases exact algorithm can be designed:

• The first phase consists in running the visibility walk algorithm with an uncertified orientation predicate; we call this phase the *relaxation phase*. This phase either terminates in some simplex and returns that simplex, or visits a number of simplices above some threshold and then returns the current simplex. This threshold mechanism avoids a possible non-termination of the relaxation phase that rounding errors may produce in pathological situations. In any case, this phase returns a simplex (hopefully) not too far from the solution.

• The second phase consists in running the visibility walk algorithm with a certified orientation predicate starting at the simplex returned by the relaxation phase, until the solution is found; we call this phase the *certification phase*. If the first phase returned the correct simplex, which is the most common case, this phase just check that simplex is the right one.

Structural filtering accelerates around 20% the point location procedure; therefore, we adopt this scheme for experiments presented in this paper. (We have implemented structural filtering in CGAL 3.8.)

3 Distribution Condition

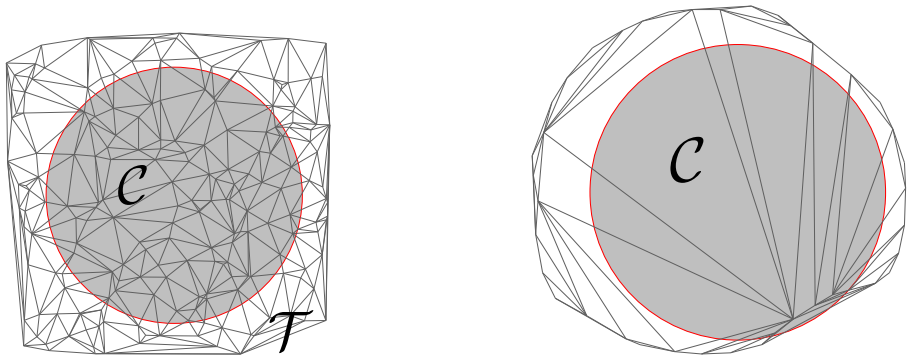
To analyze the complexity of the straight walk and derived strategies for point location, we need some hypotheses claiming that the behavior of a walk in a given region $\mathcal{C}$ of the triangulation is as follows.

> **Distribution Condition**: *Let Δ be a triangulation scheme (such as Delaunay, Regular, ...), let ρ be a distribution of points with compact support $\Sigma \subset \mathbb{R}^d$, and let $\mathcal{C}$ be a region inside Σ. For a triangulation $\mathcal{T}$ of n points following distribution ρ and built upon the triangulation scheme Δ, the Distribution Condition is satisfied if there exists a function $\mathcal{F} : \mathbb{N} \to \mathbb{R}$, such that for a segment $s \subseteq \mathcal{C}$, the expected number of simplices of $\mathcal{T}$ intersected by s, averaging on the choice of the sites in the distribution, is* ***less than****.*

One known case where the Distribution Condition is satisfied is the Delaunay triangulations of points following the Poisson distribution in dimension $d = 2$ [21, 9] and $d = 3$ [39], where $\mathcal{F}(n) = O(n^{1/d})$, for any region $\mathcal{C}$; see Figure 2(a). We believe that the distribution condition generalizes to other kinds of triangulation schemes and other kinds of distributions. An interesting case seems to be the Delaunay triangulation of points lying on some manifold in a space of dimension d ; see Figure 2(b)). The following conjecture is supported by our experiments (Section 7.1).

Conjecture 2. *The Delaunay triangulations in dimension d of n points evenly distributed on a bounded manifold Π of dimension δ, satisfy the Distribution Condition inside the convex hull of Π, with $\mathcal{F}(n) = O(n^{1/\delta})$.*

The Distribution Condition affects the relationship between the cost of locating points and the proximity between points. Let $\mathcal{T}$ be a triangulation of n points following some distribution with compact support



(a) (b)

Figure 2: **Distribution Condition.** *(a)* $\mathcal{F}(n) = O(\sqrt{n})$*, (b)* $\mathcal{F}(n) = O(n)$.

in $\mathbb{R}^d$, if the distribution condition is satisfied for a region $\mathcal{C}$ in the convex hull of $\mathcal{T}$, the expected cost of locating in $\mathcal{T}$ a finite sequence S of m query points lying in $\mathcal{C}$ by straight walk is at most

$$\mathcal{F}(n) \cdot \sum_{i=1}^{m-1} |e_i| + m, \tag{2}$$

where e_i is the line segment formed by the i-th starting point of the walk and the i-th query-point. The structure formed by all these segments has a special meaning for point location purpose. We shall see this meaning in what follows.

Let $S = \{q_1, q_2, \ldots, q_m\}$ be a sequence of queries. For a new query, the walk has to start from a point whose location is already known, i.e. a point inside a simplex visited during a previous walk. Thus the line segments e_i, $1 \leq i \leq m-1$, formed by (i) the starting point of the i-th walk toward q_i, and (ii) q_i itself, must be connected. Therefore, the graph $\mathcal{E}$ formed by these line segments e_i is a tree spanning the query points; we call such a tree the *Location Tree*. Its length is given by $\|\mathcal{E}\| = \sum_{e \in \mathcal{E}} \|e\|$. Eq.(2) can be rewritten as the following expression:

$$\mathcal{F}(n) \cdot \|\mathcal{E}\| + m. \tag{3}$$

In the next section, we review some classical trees embedded in the space and the growth rate of their length.

3.1 On Trees Embedded in $\mathbb{R}^d$

The tree theory is older than computational geometry itself. Here, we mention some of the well-known trees (and graphs) [44], which are related with the theory of point location. Let $S = \{p_i, 1 \leq i \leq n\}$ be a set of points in $\mathbb{R}^d$ and $G = (V, E)$ be the complete graph such that the vertex $v_i \in V$ is embedded on the point $p_i \in S$; the edge $e_{ij} \in E$ linking two vertices v_i and v_j is weighted by its Euclidean length $\|p_i - p_j\|$. G is usually referred to as the *geometric graph* of S.

We review below some well-known trees. Two special kinds of tree get a special name: (i) a *star* is a tree having one vertex that is linked to all others; and (ii) a *path* is a tree having all vertices of degree 2 but two with degree 1.

EMST. Among all the trees spanning S, a tree with the minimal length is called an *Euclidean minimum spanning tree* of S and denoted $EMST(S)$; see Figure 3(a). $EMST$ can be computed with a greedy algorithm at a polynomial complexity.

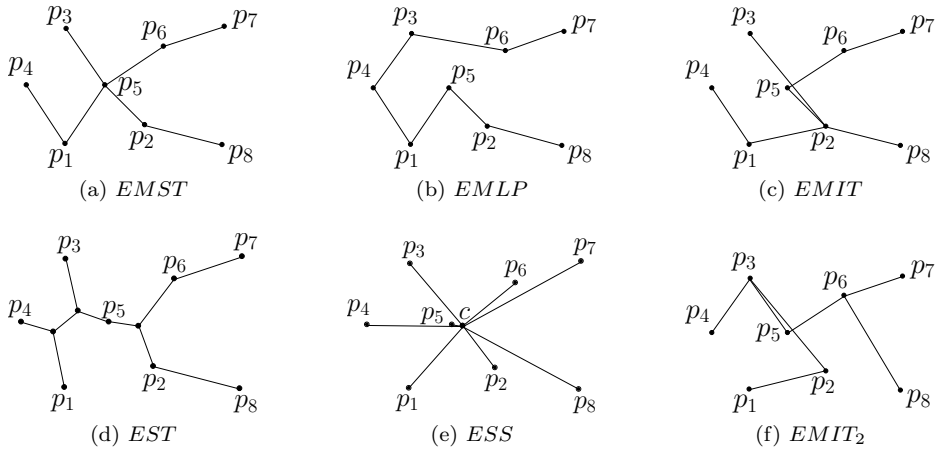


Figure 3: **Trees embedded in $\mathbb{R}^d$.**

EMLP. If instead of searching a tree, we search a path with minimal length spanning S, we get the *Euclidean minimum length path* denoted by $EMLP(S)$; see Figure 3(b). Another related problem is the search for a minimal tour spanning S: the *Euclidean traveling salesman tour*, denoted by $ETST$. Both problems are NP-hard.

Since a complete traversal of the $EMST$ (either prefix, infix or postfix) produces a tour, and removing an edge of $ETST$ produces a path, we have

$$\|EMST(S)\| \leq \|EMLP(S)\| < \|ETST(S)\| < 2\|EMST(S)\| \tag{4}$$

EMIT. Above, subgraphs of G are independent of any ordering of the vertices. Now, consider that an ordering is given by a permutation σ, vertices are inserted in the order $v_{\sigma(1)}, v_{\sigma(2)}, \ldots, v_{\sigma(n)}$. We build incrementally a spanning tree T_i for $S_i = \{p_{\sigma(j)} \in S, i \leq j\}$ with $T_1 = \{v_{\sigma(1)}\}$, $T_i = T_{i-1} \cup \{v_{\sigma(i)}v_{\sigma(j)}\}$ and a fixed k, with $1 \leq k < n$, such that $v_{\sigma(i)}v_{\sigma(j)}$ has the shortest length for any $\max(1, i-k) \leq j < i$. This tree is called the *Euclidean minimum k-insertion tree*, and is denoted by $EMIT_k(S)$; see Figure 3(f). When $k = n-1$, we write $EMIT(S)$; see Figure 3(c). $\|EMIT(S)\|$ depends on σ and for some permutations it coincides with $\|EMST(S)\|$. Unlike the previous trees, $EMIT$ and $EMIT_k$ do not require points to be known in advance, and hence they are dynamic structures.

EST. The use of additional vertices usually allows to decrease the length of a tree. Such additional vertices are called *Steiner points* and the minimum-length tree with Steiner points is the *Euclidean Steiner tree* of S; it is denoted by $EST(S)$; see Figure 3(d). Finding EST is NP-hard.

ESS. A star has one vertex linked to all other vertices. If this vertex is an additional vertex that does not belong to V, we can choose its position so as to minimize the length of the star. This point is called the *Fermat-Weber point* of S and the associated star is denoted by $ESS(S)$ (*Euclidean Steiner star*); see Figure 3(e).

3.2 On the Growth Rate of Trees in $\mathbb{R}^d$.

We present here some results on the length of the above-mentioned structures. We start by subgraphs independent of an ordering of the vertices. The Beardwood, Halton and Hammersley theorem [8] states that if p_i are i.i.d. random variables with compact support, then $\|ETST(S)\| = O(m^{1-1/d})$ with probability 1; by Eq.(4) the same bound is obtained for both $\|EMLP(S)\|$ and $\|EMST(S)\|$. While this result gives a practical bound on the complexity, they are dependent on probabilistic hypotheses; this was shown to be unnecessary. Steele and Snyder prove [46] that the complexity of these graphs remains bounded by $O(m^{1-1/d})$ even in the worst case.

Consider the length of the path formed by sequentially visiting each vertex in V. This gives a total length of $\sum_{i=2}^{m} \|p_{i-1}p_i\|$. Let $V_\sigma = \{p_{\sigma(1)}, p_{\sigma(2)}, \ldots, p_{\sigma(m)}\}$ be a sequence of m points made by reordering V with a permutation function σ such that points in V_σ would appear in sequence on some *space-filling* curve. Platzman and Bartholdi [41] proved that in two dimensions the length of the path made by visiting S_σ sequentially is a $O(\log m)$ approximation of $\|ETST(S)\|$, and hence $\sum_{i=2}^{m} \|p_{\sigma(i-1)}p_{\sigma(i)}\| = O(\sqrt{m}\log m)$. One of the main interests of such heuristic is that σ can be found in $O(m \log m)$ time.

Finally, Steele shows that the growth rate of $\|EMIT(S)\|$ is as large as the one of $\|EMST(S)\|$ [45] (this result has been recently generalized [13]). More precisely:

Theorem 3 (Steele [45])**.** *Let S be a sequence of m points in $[0,1]^d$, $d \geq 2$, then we have the following inequality: $|EMST(S)| \leq |EMIT(S)| \leq \gamma_d m^{1-1/d}$. Here, $\gamma_d = 1 + 2^{4d} d^{d/2}/(d-1)$ is a constant depending only on d.*

4 Constant-Size-Memory Strategies

In this section, we analyze the length of the Location Tree of strategies that store a constant number of possible starting points for a straight walk; this is naturally followed by a comparative study between them. Proofs of Theorems 4, 5, and 11 below are restricted cases of theorems proved in a companion paper [13].

4.1 Fixed-point strategy.

In the fixed-point strategy, the same point c is used as starting point for all the queries, then the Location Tree is the star rooted at c, denoted by $S_c(S)$. The best Location Tree we can imagine is the Steiner star, but of course computing it is not an option, neither in a dynamic setting nor in a static setting. This strategy is used in practice: In CGAL 3.6, the default starting point for a walk is the so-called *vertex at infinity*; thus the walk starts somewhere on the convex hull, which looks like a kind of worst strategy.

In the worst case, one can easily find a set of query-points S such that $|ESS(S)| = \Omega(m)$, or such that $|S_c(S)|/|ESS(S)|$ goes to infinity for some c. Now we focus on the case of evenly distributed queries.

Theorem 4 ([13], for $\alpha = 1$)**.** *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the expected length of the Location Tree of the best fixed-point strategy is*

$$\left(\frac{d}{d+1}\right) \cdot m.$$

Theorem 5 ([13], for $\alpha = 1$)**.** *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the expected length of the Location Tree of the worst (on the choice of c inside the ball) fixed-point strategy is*

$$2^{d+1}\left(\frac{2d+1}{2d+2}\right)\frac{B\left(\frac{d}{2}+\frac{1}{2},\frac{d}{2}+1\right)}{B\left(\frac{d}{2}+\frac{1}{2},\frac{1}{2}\right)} \cdot m,$$

where $B(x,y) = \int_0^1 \lambda^{x-1}(1-\lambda)^{y-1}d\lambda$ is the so-called Beta function.

Corollary 6. *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the ratio between the expected lengths of the Location Tree of the best and worst fixed-point strategies is at most 2 (for $d = 1$), and at least $\sqrt{2}$ (when $d \to \infty$).*

Figure 4 gives the expected average length of an edge of the best and worst fixed-point Location Trees.

4.2 Last-point strategy.

An alternative to the fixed-point strategy is the last-point strategy. To locate a new query-point, the walk starts from the previously located query. The Location Tree obtained with such a strategy is a path. When $\mathcal{T}$ satisfies the Distribution Condition, the optimal path is the $EMLP(S)$.

In the worst case, the length of such a path is clearly $\Omega(m)$; a straightforward example is to repeat alternatively the two same queries. In contrast with the fixed-point strategy, the last-point strategy depends on the query distribution. If the queries have some spatial coherence, it is clear that we improve on the fixed-point strategy. Such a coherence may come from the application, or by reordering the queries. There is always a permutation of indices on S such that the total length of the path is sub-linear [46, 25]. Furthermore, in two dimensions, one could find such permutation in $O(m \log m)$ time complexity [41].

Now, the question we address is "if there is no spatial coherence, how the fixed and last point strategies do compare?".

Theorem 7. *The ratio between the lengths of the Location Tree of the last-point strategy and the fixed-point strategy is at most* 2.

Proof. This is an easy consequence of the triangle inequality. Take $S = p_1, \ldots, p_m$, and any fixed-point c. Then we have:

$$|p_i p_{i+1}| \leq |cp_i| + |cp_{i+1}|,$$

for all $1 \leq i < m$. Summing the term above for each value of i leads to the inequality:

$$\sum_{i=1}^{m-1} |p_i p_{i+1}| \leq \sum_{i=1}^{m-1} |cp_i| + \sum_{i=2}^{m} |cp_i| \leq 2 \sum_{i=1}^{m} |cp_i|,$$

which completes the proof. □

Theorem 8. *The ratio between the lengths of the Location Tree of the last-point strategy and the fixed-point strategy is arbitrarily small.*

Proof. Consider a set of m queries distributed on a circle in $\mathbb{R}^d$. If the queries are visited along the circle, the length of the location tree of the last-point strategy is $O(1)$, while $|ESS| = \Omega(m)$. □

Combining the results in Theorem 7 and Theorem 8, it is reasonable to conclude that the last-point strategy is better in general, as the improvement the fixed-point strategy could bring does not pay the price of its worst-case behavior. We now study the case of evenly distributed queries.

Theorem 9. *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the expected length of the Location Tree of the last-point strategy is*

$$2^{d+1} \left(\frac{d}{d+1} \right) \frac{B\left(\frac{d}{2} + \frac{1}{2}, \frac{d}{2} + 1\right)}{B\left(\frac{d}{2} + \frac{1}{2}, \frac{1}{2}\right)} \cdot m.$$

Proof. This is equivalent to find the expected length of a random segment determined by two points uniformly independent and identically distributed in the unit ball, which is given in [42]. □

Theorems 4, 5, and 9 give the following result:

Corollary 10. *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the ratio between the expected lengths of the Location Tree of the last-point and the best fixed-point strategies is at most $\sqrt{2}$ (when $d \to \infty$), and at least $4/3$ (when $d = 1$) whereas the ratio between the expected lengths of the Location Tree of the last-point and the worst fixed-point strategies is $2d/(2d+1)$.*

As shown in Figure 4, the last-point strategy is in between the best and worst fixed-point strategies, but closer and closer to the worst one when d increases. Thus, in the context of evenly distributed points in a ball, the last-point strategy cannot be worse than any fixed point strategy by more than a factor of $\sqrt{2}$. Still, the fixed-point strategy may have some interests under some conditions: (i) queries are known *a priori* to be random **and**; (ii) a reasonable approximation of the center of $ESS(S)$ can be found.

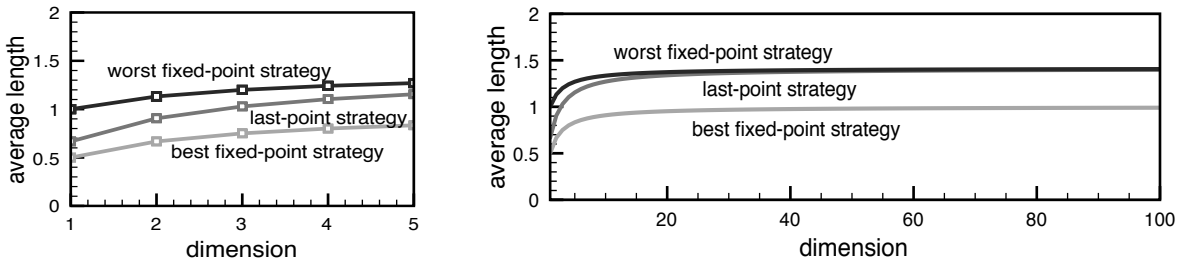


Figure 4: **Average lengths.** *Average lengths of an edge of the last-point, best and worst fixed-point Location Trees. The domain $\mathcal{C}$ here is a d-dimensional ball, and the queries are evenly distributed in $\mathcal{C}$.*

4.3 k-last-points strategy.

We explore here a variation of the last-point strategy. Instead of remembering the place where the last query was located, we store the places where the k last queries were located, for some small constant k. These k places are called *landmarks* in what follows. Then to process a new query, the closest landmark are determined by $O(k)$ brute-force comparisons, then a walk is performed from there. This strategy has some similarity with Jump & Walk, the main differences are that the sample has fixed size and depends on the query distribution (it is dynamically modified).

The Location Tree associated with such a strategy is $EMIT_k$. It has bounded degree $k+1$ (or the *kissing number* in dimension d, if it is smaller than $k+1$) and its length is greater than $\|EMST\|$ and smaller than the length of the path associated to the same vertices ordering, thus previous results provide upper and lower bounds. A tree of length $\Omega(m/k) = \Omega(m)$ is easily achieved by repeating a sequence of k queries along a circle of length 1. The following Theorem gives the complexity when the queries are evenly distributed:

Theorem 11 ([13], for $\alpha = 1$)**.** *Let S be a sequence of m query-points uniformly independent and identically distributed in the unit ball, then the expected length of the Location Tree of the k-last-points strategy satisfies*

$$\left(\frac{1}{d}\right) B\left(k+1, \frac{1}{d}\right) \cdot m \leq E(length) \leq 2\left(\frac{1}{d}\right) B\left(k+1, \frac{1}{d}\right) \cdot m. \tag{5}$$

Remark. Note that the constant γ_d in Theorem 3 looks big. But it is indeed too pessimistic. Theorem 11 leads to Theorem 12, which gives a better constant for queries evenly distributed inside the unit sphere.

Theorem 12. *Let S be a sequence of k query-points uniformly independent and identically distributed in the unit ball, then the expected value of $\|EMIT(S)\|$ is $2 \cdot \Gamma(1+1/d) \cdot k^{1-1/d}$ as $k \to \infty$, where $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\mathrm{d}t$ is the Gamma function.*

Proof. From Theorem 11, we have that the expected length l_i of the i-th edge of $\|EMIT(S)\|$ in the unit ball for evenly distributed points, is such that $l_i \leq \left(\frac{2}{d}\right) B\left(i+1, \frac{1}{d}\right)$. Where $B(x,y) = \int_0^1 \lambda^{x-1}(1-\lambda)^{y-1}\mathrm{d}\lambda$ is the Beta function. Summing the expression above for $k-1$ edges, and using the Stirling's identity $B(x,y) \sim \Gamma(y)x^{-y}$, we have that there exists $k_0 < \infty$, such that for $k > k_0$, $\|EMIT(S)\|$ is bounded by $(1+\epsilon) \cdot 2 \cdot \Gamma(1+1/d) \cdot k^{1-1/d}$ with ϵ as small as we want. □

Intuitively, if the queries have some not too strong spatial coherence, the k-last-points strategy seems a good way to improve the last-point strategy. Surprisingly, experiments in Section 7 shows that even if the points have some strong coherence, a small k strictly greater than 1 improves on the last-point strategy when points are sorted along a space-filling curve. More precisely, $k = 6$ improves the location time by up to 30% on some data sets. Experiments were done using the **stochastic walk**.

5 Keep, Jump & Walk

5.1 Preliminaries: Jump & Walk

The Jump & Walk strategy takes a random sample of k vertices (the dots in Figure 5) of $\mathcal{T}$, and uses a two-steps location process in order to locate a query q. First, the jump step determines the nearest vertex in the sample in (brute-force) $O(k)$ time, then a walk in $\mathcal{T}$ is performed from that vertex. In the classical analysis of Jump & Walk, it is assumed that $\mathcal{T}$ is the Delaunay triangulation of points evenly distributed; taking $k = n^{1/(d+1)}$ gives a complexity of $O(n^{1/(d+1)})$ [39, 22].

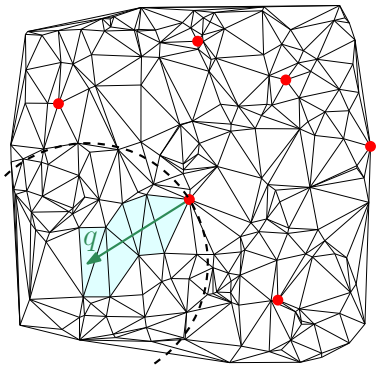


Figure 5: **Jump & Walk.** *The walk starts from the nearest landmark (represented by dots above) with respect to the query. Then it ends at the cell containing the query.*

5.2 A distribution-sensitive modification: Keep, Jump, & Walk

As mentioned before, the Jump & Walk strategy [39, 22] uses a set of k landmarks randomly chosen amongst the vertices of $\mathcal{T}$, then a query is located by walking from the closest landmark. In order to ensure adaptation to the query distribution, instead of using vertices of $\mathcal{T}$ as landmarks, we *keep* previous queries as landmarks. Then, we have several possibilities: (i) we can use k queries chosen at random in previous queries, (ii) we can use the k last queries for the set of landmarks, and (iii) we can keep all the queries as landmarks, and regularly clear the landmarks set after a batch of k queries

For any rule to construct the set of landmarks, the time to process a query q splits in:
— Keep: the time $K(k)$ for updating the set of landmarks if needed,
— Jump: the time $J(k)$ for finding the closest landmark l_q, and
— Walk: the time $W(k)$ to walk from l_q to q.

This modification performs surprisingly well in practice, experimental results for method (ii) are provided in Section 7.4.

We analyze in this section the Keep, Jump, & Walk strategy considering the **straight walk** as the walk strategy. We start with the following lemma, which is a fundamental piece in this section.

Lemma 13. *Let $\mathcal{T}$ be a triangulation of n points following some distribution with compact support in $\mathbb{R}^d$, if the Distribution Condition is satisfied for a region $\mathcal{C}$ in the convex hull of $\mathcal{T}$, then $W(k)$ has an expected amortized $O\left(\mathcal{F}(n) \cdot k^{-1/d} + 1\right)$ complexity for k queries.*

Proof. For k queries, the Location Tree of each variation above is an $EMIT$ with k vertices. Let $\mathcal{T}$ be a triangulation of n points following some distribution with compact support in $\mathbb{R}^d$, if the Distribution Condition is satisfied for a region $\mathcal{C}$ in the convex hull of $\mathcal{T}$, the expected cost of locating in $\mathcal{T}$ a finite

sequence S of k query points lying in $\mathcal{C}$ is, from Eq.(3), at most

$$\mathcal{F}(n) \cdot \sum_{e \in \mathcal{E}} \|e\| + k = \mathcal{F}(n) \cdot \|\mathcal{E}\| + k = O\left(\mathcal{F}(n) \cdot k^{1-1/d} + k\right), \tag{6}$$

since, by Theorem 3, $\|EMIT\| = O(k^{1-1/d})$. Therefore, $W(k)$ has an expected amortized $O\left(\mathcal{F}(n) \cdot k^{-1/d} + 1\right)$ complexity for k queries. □

Combining various options for $\mathcal{F}(n)$ and the data-structure to store the landmarks, gives us some interesting possibilities. The trick is always to balance these different costs, since increasing one decreases another.

Jump & Walk. Classical Jump & Walk uses a simple data-structure (e.g. a list) to store the random sample of $\mathcal{T}$ and assumes $\mathcal{F}(n) = O(n^{1/d})$. Here, we use the same data-structure to store the set of landmarks. Keep step decides whether the query is kept at a landmark and inserts it if needed. This takes $K(k) = O(1)$. Jump step takes $J(k) = O(k)$. Then, using Lemma 13 and taking $k = n^{1/(d+1)}$ landmarks amongst the queries ensures an amortized query time of $O(n^{1/(d+1)})$. It is noteworthy that the complexity obtained here matches the classical Jump & Walk complexity with no hypotheses on the distribution of query-points (naturally, the queries must lie in the region $\mathcal{C}$, which in turn must lie inside the domain of $\mathcal{T}$, see Section 3).

Outside this classical framework, Jump & Walk has some interests, even with weaker hypotheses. In general considering Lemma 13, taking $k = \mathcal{F}(n)^{1-1/(d+1)}$ balances the jump and the walk costs. Another remark is that if the landmarks are a random subset of the vertices of $\mathcal{T}$ (as is the classical Jump & Walk), then the cost of the walk is $\mathcal{F}(n/k)$ [19, Variation of Lemma 4]. Assuming $\mathcal{F}(j) = O(j^\beta)$, the jump and the walk costs are balanced by taking $k = n^{1-1/(\beta+1)}$ in this case.

Besides, if Conjecture 2 is verified, Keep, Jump, & Walk should use a sample of size

$$k = O\left(\left(n^{1/(d-1)}\right)^{1-1/(d+1)}\right)$$

to construct Delaunay triangulation for points on a hypersurface, and not $O(n^{1/(d+1)})$ as for random points in the space. In particular, k should be $O(n^{2/3})$ in 2D, and $O(n^{3/8})$ in 3D; this is verified experimentally in Section 7.3, and should be applied in surface reconstruction applications.

Despite Jump & Walk's non-optimality, for up to few millions of points, if the number of landmarks is carefully chosen, this strategy might be actually competitive; see Section 7.3, and Section 7.4.

Walk & Walk. In Walk & Walk, the data-structure to store the landmarks is a Delaunay triangulation $\mathcal{L}$, in which it is possible to walk (notice that $\mathcal{T}$ may not be a Delaunay triangulation). Assuming a random order on the landmarks, inserting or deleting a landmark after location takes $K(k) = O(1)$ and jump step takes $J(k) = O(\mathcal{F}(k))$.

If the queries and the sites are both evenly distributed we get $J(k) = O(k^{1/d})$ and, by Lemma 13, $W(k) = k^{-1/d} \cdot \mathcal{F}(n) = O(k^{-1/d} \cdot n^{1/d})$, which gives $k = \sqrt{n}$ to balance the jump and walk costs. Finally, the point location takes expected amortized time $O(n^{1/2d})$.

If walking inside $\mathcal{T}$ and $\mathcal{L}$ takes linear time, $k = n^{1-1/(d+1)}$ balances Walk & Walk costs.

Delaunay Hierarchy of Queries. A natural idea is to use several layers of triangulations, walking at each level from the location at the coarser layer. When the landmarks are vertices of $\mathcal{T}$ and each sample takes a constant ratio of the vertices at the level below, this idea yields the Delaunay hierarchy [19].

We can handle *bad* triangulations by looking at the tradeoff between query time and storage. If the region $\mathcal{C}$ of $\mathcal{T}$ is such that $\mathcal{F}(n)$ is a polynomial on n, and queries are well-distributed, we can get interesting query time once a polynomial storage is allowed. More precisely, if the queries are such that a random sample of the queries has a Delaunay triangulation of expected linear size (always true in 2D), then using a random sample of k queries for the landmarks and a Delaunay hierarchy to store $\mathcal{L}$, gives $K(k) = J(k) = O(\log k)$. Then by Lemma 13 we have $W(k) = O(k^{-1/d} \cdot \mathcal{F}(n))$ (amortized) and taking $k = \mathcal{F}(n)^d / \log^d n$ balances jump and walk costs, leading to an expected amortized logarithmic query time.

The conditions such that Walk & Walk and the Delaunay Hierarchy of Queries are interesting are quite restrictive on n and on the discrepancy between the sites and queries distribution, thus they are mentioned here but they are not tried in the benchmarks section.

6 Climbing Up in the Delaunay Hierarchy

In this section, we show how the Delaunay hierarchy can be made distribution-sensitive under some hypotheses. Assume $\mathcal{T}$ is a Delaunay triangulation, then classical use of the Delaunay hierarchy provides a logarithmic cost in the total size of $\mathcal{T}$ to locate a point. The cost we reach here is logarithmic in the number of vertices of $\mathcal{T}$ *in between* the starting point and the query.

Given a set of n points $\mathcal{P}$ in the space, we assume that the expected size of the Delaunay triangulation of a random sample of size r of $\mathcal{P}$ has linear size. The hypothesis always holds in 2D, and is proved in several other situations: points randomly distributed in space [23] or on a hypersurface [26, 6, 7, 4]. The Delaunay hierarchy [19] constructs random samples $\mathcal{P} = \mathcal{P}_0 \supseteq \mathcal{P}_1 \supseteq \mathcal{P}_2 \supseteq \ldots \supseteq \mathcal{P}_h$ such that $Prob(p \in \mathcal{P}_{i+1} \,|\, p \in \mathcal{P}_i) = 1/\alpha$ for some constant $\alpha > 1$. The $h+1$ Delaunay triangulations $\mathcal{D}_i$ of $\mathcal{P}_i$ are computed and the hierarchy is used to find the nearest neighbor of a query q by walking at one level i from the nearest neighbor of q at the level $i+1$. It is proven that the expected cost of walking at one level is $O(\alpha)$ and since the expected number of levels is $\log_\alpha n$, we obtain a logarithmic expected time to descend the hierarchy for point location.

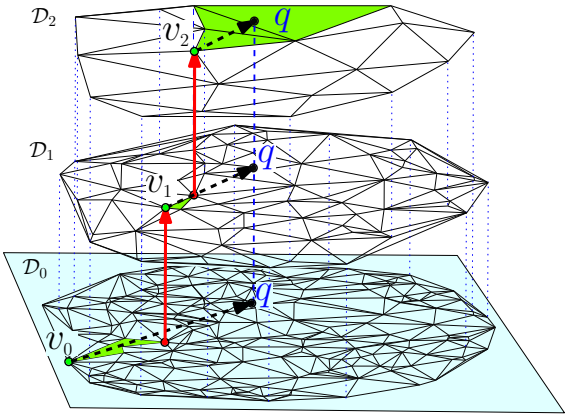


Figure 6: **Climbing.**

If a good starting vertex $v = v_0$ in $\mathcal{D}_0$ is known, the Delaunay hierarchy can be used in another way: From v_0 a walk starts in $\mathcal{D}_0$ visiting simplices crossed by segment v_0q; the walk is stopped, either if the simplex containing q is found, or if a simplex having a vertex v_1 belonging to the sample $\mathcal{P}_1$ is found. If the walk stops because v_1 is found, then a new walk in $\mathcal{D}_1$ starts at v_1 along segment v_1q. This process continues recursively up to the level l, where a simplex of $\mathcal{D}_l$ that contains q is found; see Figure 6. Finally, the hierarchy is descended as in the usual point location. Theorem 14 bounds the complexity of this procedure.

Theorem 14. *Given a set of n points $\mathcal{P}$, and a convex region $\mathcal{C} \subseteq CH(\mathcal{P})$, such that the Delaunay triangulation of a random sample of size r of $\mathcal{P}$*
(i) has expected size $O(r)$,
(ii) satisfies the Distribution Condition in $\mathcal{C}$ with $\mathcal{F}(r) = O(r^\beta)$ for some constant β,
then the expected cost of climbing and descending the Delaunay hierarchy from a vertex v to a query point q, both lying in $\mathcal{C}$, is $O(\log w)$, where w is the expected cost of the walk from v to q in $\mathcal{D}$ the Delaunay triangulation of $\mathcal{P}$.

Proof. **Climbing one level.** Since the probability that any vertex of $\mathcal{D}_i$ belongs to $\mathcal{D}_{i+1}$ is $1/\alpha$, and that each time a new simplex is visited during the walk a new vertex is discovered, the expected number of visited simplices before the detection of a vertex that belongs to $\mathcal{D}_{i+1}$ is at most $1 + \sum_{j=0}^{\infty} j \frac{1}{\alpha} \left(1 - \frac{1}{\alpha}\right)^j = \alpha$.
Descending one level. The cost of descending one level is $O(\alpha)$ [19, Lemma 4].
Number of levels. Let w_i denote the number of edges crossed by v_iq in $\mathcal{D}_i$; the Distribution Condition gives $w_i = \mathcal{F}(n/\alpha^i)\|v_iq\| \leq \mathcal{F}(n/\alpha^i)\|v_0q\|$. If $\mathcal{F}(r)$ is a polynomial function $O(r^\beta)$, the expected number of

levels that we climb before descending is less than $l = (\log w_0)/\beta$, since we have

$$\begin{aligned} w_l &= \mathcal{F}(n/\alpha^l)\|v_l q\| \leq \mathcal{F}(n/\alpha^l)\|v_0 q\| \\ &\leq w_0/\alpha^{l\beta} = w_0/\alpha^{\log w_0} = 1 \end{aligned}$$

(where the big O have been omitted). Then, at level l the walk takes constant time. □

Remind that in Section 5, we keep landmarks in order to improve the classical walking algorithm, which leads to Keep, Jump, & Walk. Now, it is natural to improve the climbing algorithm described above by adding landmarks in $\mathcal{D}_0$ as well, and starting the climbing procedure from a good landmark. Since the complexity of climbing is $O(\log w)$, to balance the different costs, the number of considered landmarks has also to be $O(\log w)$. Using the hypotheses in Theorem 14, this number is given by $O(\log w) = O(\log(\mathcal{F}(n) \cdot \|pq\| + 1)) = O(\log(n\|pq\|))$; neither $\mathcal{F}(n)$ nor the constant in the big O need to be known to ensure such a complexity. This approach is evaluated experimentally in the next section.

7 Experimental Results

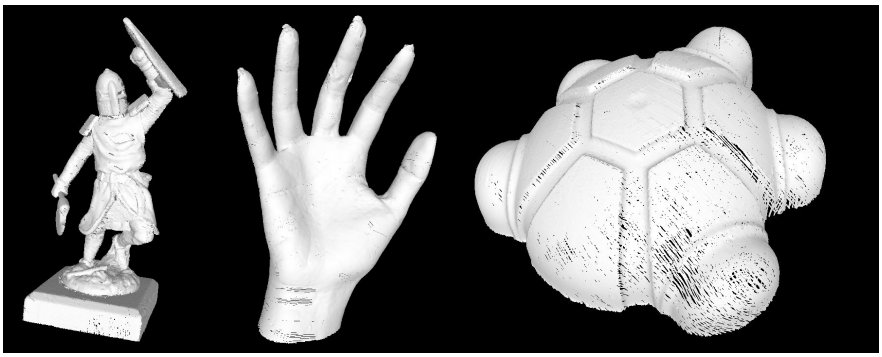


Figure 7: **Scanned models.**

Experiments have been realized on synthetic and realistic models (scaled to fit in the unit cube). The scanned models used here: **Galaad**, **Pooran's hand**, and **Toy Turtle** are taken from the Aim@shape repository; see Figure 7. The hardware used for the experiments described in the sequel, is a MacBook Pro 3,1 equipped with an 2.6 GHz Intel Core 2 processor and 2 GB of RAM, Mac OS X version 10.5.7. The software uses CGAL 3.6 and is compiled with `g++ 4.3.2` and options `-O3 -DNDEBUG`. All the triangulations in the experiments are Delaunay triangulations. Each experiment was repeated 30 times, and the average is taken. We remind the reader that the walking strategy used in the section is the stochastic walk.

We consider data sets in 2D and 3D; they are described in Table 2.

Files of different sizes, smaller than the original model are obtained by taking random samples of the main file with the desired number of points.

7.1 The Distribution Condition

Our first set of experiments is an experimental verification of the Distribution Condition. We compute the Delaunay triangulation of different inputs, either artificial or realistic, with several sizes; for realistic inputs we construct files of various sizes by taking random samples of the desired size.

Figure 8 shows the number of crossed tetrahedra in terms of the length of the walk, for various randomly chosen walks in the triangulation. A linear behavior with some dispersion is shown. Though, from this

uniform square (2D)	Points evenly distributed in the unit square.
anisotropic square (2D)	Points distributed in a square with a $\rho = x^2$ density.
ellipse (2D)	Points evenly distributed on an ellipse; the lengths of the ellipsoid axes are 1/2, and 1.
uniform cube (3D)	Points evenly distributed in the unit cube.
anisotropic cube (3D)	Points distributed in a cube with a $\rho = x^2$ density.
ellipsoid (3D)	Points evenly distributed on the surface of an ellipsoid; the lengths of the ellipsoid axes are 1/3, 2/3, and 1.
cylinder (3D)	Points evenly distributed on the surface of a cylinder; height and diameter of the cylinder are unitary.
Galaad, **Pooran's hand**, and **Toy Turtle** (3D)	a 3D model of a physical object.

Table 2: **Data sets in 2D and 3D.**

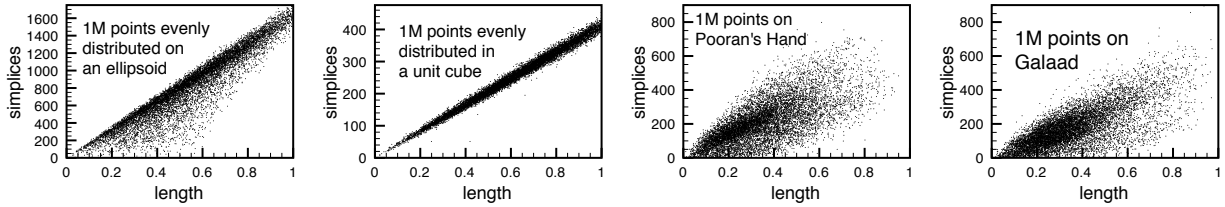


Figure 8: **Distribution condition.** ♯ *of crossed tetrahedra in terms of the length of the walk. The number of points sampled in each model here is* 2^{20}.

experiment, the walks that deviates significantly from the average behavior are more likely to be faster than slower, which is a good news

From Figure 8, the slope of lines best fitting these point clouds give an estimation of $\mathcal{F}(n)$ for a particular n (namely $n = 2^{20}$). By doing these computations for several n, we draw $\mathcal{F}(n)$ in terms of the triangulation size in Figure 9.

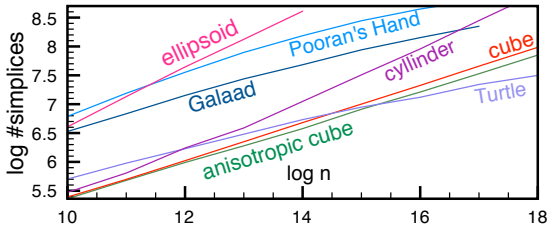
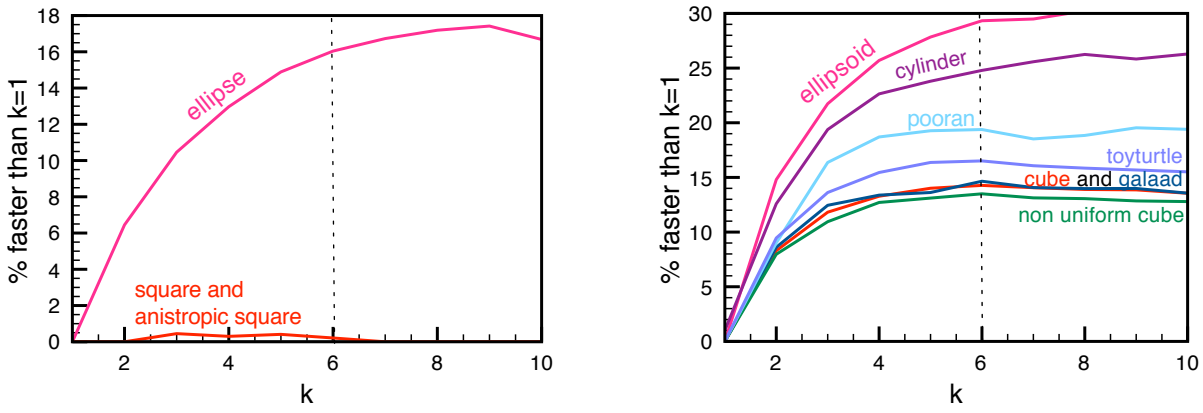


Figure 9: $\mathcal{F}(\mathbf{n})$ **for some interesting cases.** *If* $\mathcal{F}(n)$ *is* n^β, β *is given by the slope of above "lines" .*

If $\mathcal{F}(n)$ is clearly bounded by a polynomial on n, then curves in Figure 9 should lie below some line. Now, from the biggest slope of lines tangent to these different curves, we evaluate the exponent of n. The points sampled on an ellipsoid give $\mathcal{F}(n) \sim n^{0.52}$, and on a closed cylinder give $\mathcal{F}(n) \sim n^{0.51}$, which are not far from Conjecture 2 that claims $\mathcal{F}(n) = O(n^{1/2})$. The points evenly distributed in a cube give $\mathcal{F}(n) \sim n^{0.31}$, which is not far from $\mathcal{F}(n) = O(n^{1/3})$. For the scanned models, the curves are a bit concave, with a slope always smaller than 0.5; the Conjecture 2 is also verified in these cases, since the Distribution Condition claims only an upper bound and not an exact value for the number of visited tetrahedra.

7.2 k-last-points Strategy

CGAL library [40] uses spatial sorting [16] to introduce a strong spatial coherence in a set of points. For several models, we locate $1M$ queries evenly distributed inside the model with the k-last-point strategy after a spatial sorting of the queries. Surprisingly, using a small k slightly improves on $k = 1$ which indicates that even with such a strong coherence, k-last-points strategy is relevant. Figure 10 shows the running times on various sets for different values of k, taking $k = 6$ always improves on $k = 1$ and in some cases by a substantial amount.



(a) 2D (b) 3D

Figure 10: **Static point location with space-filling heuristic plus last-k-points strategy.**

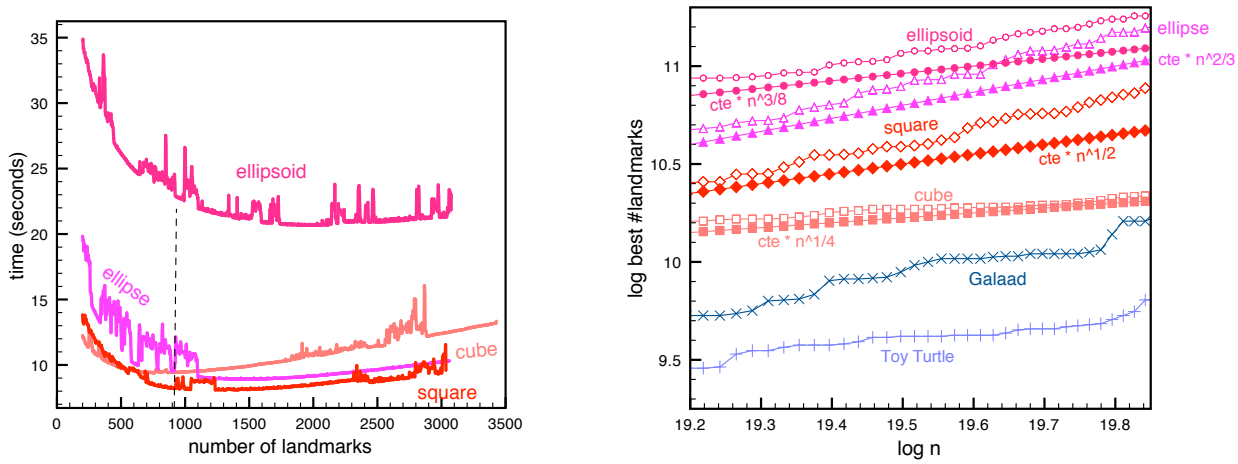
7.3 Fine-tuning Constants

Regarding Keep, Jump, & Walk strategy, in Section 5.2, we have seen that, in theory, assuming the Distribution Condition, the correct number of landmarks for random points on 2D curves and 3D surfaces should be $O(n^{2/3})$, and $O(n^{3/8})$ respectively, and not $O(n^{1/3})$ and $O(n^{1/4})$, as customary. In this section, we verify experimentally that our complexities are better estimates for uniformly random points distributed on curves and surfaces.

For each input model, both 2D and 3D, we have performed Keep, Jump, & Walk in order to locate $1M$ queries evenly distributed inside their respective Delaunay triangulations. We varied the amount of landmarks from few hundreds to around three thousands of them, and the size of the Delaunay triangulation from around $600K$ to $1M$ vertices; some of the graphs are depicted in Figure 11(a). From these graphs, we have taken the best performances, in order to showcase how the best number of landmarks varies with the number of vertices in the triangulation. We plot a log-curve so the slope of the fitted lines represent the degree of the polynomials inside the Big-O. The result is depicted in Figure 11(b). We can see that, for our input models, $O(n^{2/3})$, and $O(n^{3/8})$ are indeed better estimates for the number of landmarks in 2D curves and 3D surfaces, than $O(n^{1/3})$, and $O(n^{1/4})$ respectively.

From Figure 11(a), we can see that the performance of Keep, Jump, & Walk is harmed when too few or too much landmarks are taken. However, clearly, the extent of that damage is the greatest when too few landmarks are chosen; this is because evaluating distances is faster than evaluating orientation tests. Taking that into consideration, for the rest of the experiments, we have chosen constants inside the Big-O as indicated in Table 3.

Beside the four cases in Table 3, we also consider a fifth case: Keep, Jump, & Walk with a constant number of one hundred landmarks. When it is known that input queries have great spatial coherence, there



(a) 2D (b) 3D

Figure 11: **Estimates for the best number of landmarks in Keep, Jump, & Walk strategy.** *In the right figure, filled symbols represent the log-curve of some polynomial on n specially chosen for better depiction.*

curves (2D)	$\#\text{landmarks} = 0.23 * n^{2/3}$
volumes (2D)	$\#\text{landmarks} = 20 * n^{1/3}$
surfaces (3D)	$\#\text{landmarks} = 10 * n^{3/8}$
volumes (3D)	$\#\text{landmarks} = 42 * n^{1/4}$

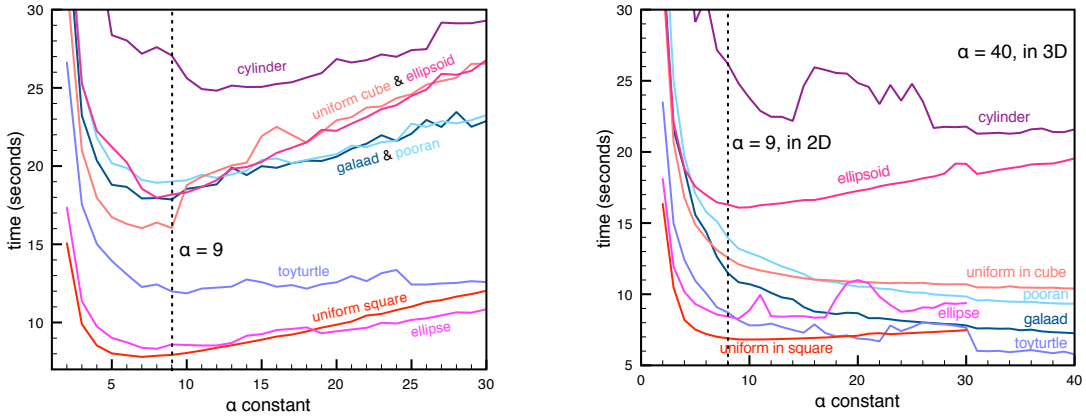
Table 3: **Our choices for the exact number of landmarks.** *These formulas are taken for models with more than around 600K vertices.*

is no need for too much landmarks; we include the later so we can have an idea of how much can be gained in performance when there is *a priori* knowledge of the spatial coherence of the queries.

Similar experiments have been done for the Climb strategy, varying the α constant, for $1M$ location queries in $1M$-sized models. The results showed in Figure 12(a) suggests that $\alpha = 9$ is a consistently good choice for the climb strategy. Contrast with the classical Delaunay hierarchy strategy, which uses $\alpha = 30$ [19] (the best constant for the uniform distribution). For the Keep, Jump, & Climb strategy, we also have varied the number of landmarks. As in the Keep, Jump, & Walk strategy, our experiments showed that too few landmarks can significantly harm the performance. However, there are no relevant gain or loss in performance by adding more landmarks than a few hundreds. In practice, we use a constant number of 500 landmarks for the Keep, Jump, & Climb strategy in both 2D and 3D, instead of the more complex procedure described at the end of Section 6; this number of landmarks behave well in practice for $1M$-sized input models. Figure 12(b) shows that, in 2D, $\alpha = 9$ remains a good choice for the Keep, Jump, & Climb strategy, however in 3D, $\alpha = 40$ becomes considerably better for almost all input models; these are also our choices for experiments in Section 7.4.

7.4 Performances

Now, we compare the performance of various point location procedures in 3D: classical Jump & Walk (**J&W**), walk starting at the previous query (**last-point**), Keep, Jump, & Walk described in Section 5 (**K&J&W**), and its variant with fewer landmarks (**few K&J&W**) as mentioned in Section 7.3, descending the Delaunay



(a) Climb

(b) Keep, Jump, & Climb

Figure 12: **Estimates for the best constants in hierarchy based strategies.** *Performances with respect to the Climb strategy, and the Keep, Jump, & Climb strategy are depicted in (a) and (b) respectively. 2D experiments don't go beyond $\alpha = 30$, and in (b) they are shown in grey.*

hierarchy (classical [19]), climbing the Delaunay Hierarchy (**Climb**), and Keep, Jump, & Climb (**K&J&C**) [2]. For this purpose we consider the following experiment scenarios.

Scenario I — This scenario is designed to show how the proximity of queries relates to the point location algorithms performance. Let $\mathcal{M}$ be an input model with 2^{20} vertices inside the unit cube, we first define S_i, the 2^i vertices of $\mathcal{M}$ closest to some chosen point $\hat{\mathbf{p}}$ in the cube. We have chosen $\hat{\mathbf{p}}$ so the radii of the smallest balls containing S_i, for $i = 15, \ldots, 20$, are not extremely discrepant. Point $\hat{\mathbf{p}}$ is $(1/10, 1/2, 1/2)$ for Pooran's hand model, $(1/4, 1/4, 7/10)$ for Galaad model, $(7/10, 3/10, 1/4)$ for Toy Turtle model, and finally $(1/2, 1/2, 1/2)$ for uniform cube. When i is large (resp. small), points are distributed in a large (resp. small) region on $\mathcal{M}$. Then, we form the sequence A_i of 2^{20} points by taking 2^{20} random point inside the smallest ball containing S_i; the value i is called the *sparsity* of the set. Figure 13 shows the computation times for point location and different strategies in function of i.

Scenario II — This scenario is designed to show how the spatial coherence of queries relates with the point location algorithms performance. Imagine a scenario where N random walkers $w_0, w_1, \ldots w_{N-1}$, are walking simultaneously with the same speed inside the unit cube containing $\mathcal{M}$, and at each steps, queries are issued for each walker position. Each random walker starts at different positions and with different directions. One step consists of a displacement of length 0.01 for all walkers. We compute the time to complete all 2^{20} queries generated by 1 to 20 random walkers in Figure 14, and 100 to 2000 random walkers in Figure 15. One single walker means a very strong spatial coherence. Conversely, several walkers mean a weaker spatial coherence; at 2000 walkers, the number of random walkers surpasses the number of landmarks for all strategies.

The walk strategy used in the experiments is the **stochastic walk**. To guarantee honest comparisons, we use the same stochastic walk implementation for all the experiments: the stochastic walk implemented in Cgal [48, 40].

From Scenario I. Figure 13 shows the total running time for answering 2^{20} queries. It can be observed that the Keep, Jump, & Walk strategy actually benefits from the proximity of the queries and is clearly better than Jump & Walk and even better than the Delaunay hierarchy for the realistic models. Keep, Jump, & Walk with few landmarks performs better than Keep, Jump, & Walk with a higher number of landmark for strong locality, which is not surprising as computing distances between a point and landmarks is not for free. However, its performance suffers a lot more when the sparsity reaches its maximum level. Naturally,

[2] All these point location strategies are also implemented in a javascript demo; we made it available at [14].

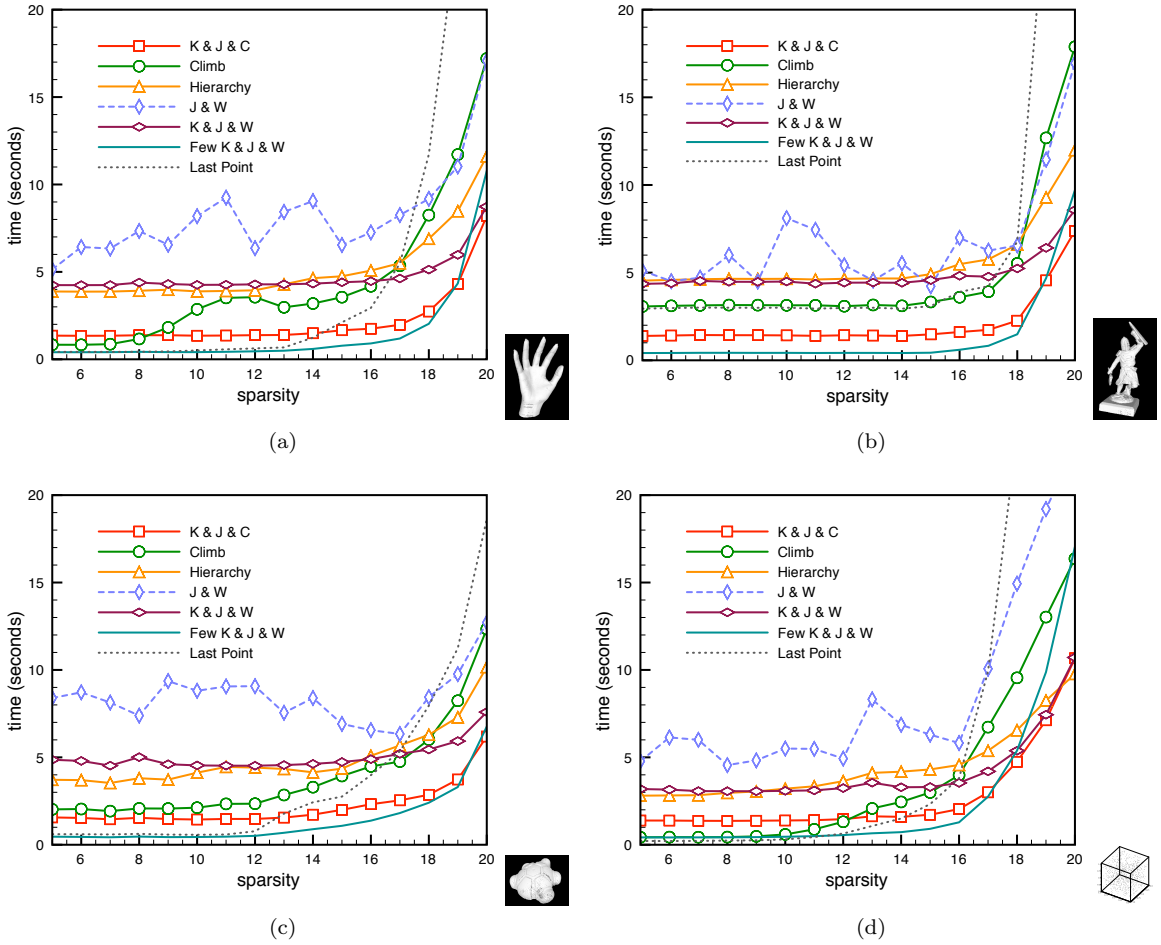


Figure 13: **Results for scenario I (proximity of queries).** *Computation times of the various algorithms in function of the sparsity of the set of points: (a)* Pooran's hand *model; (b)* Galaad *model; (c)* Toy Turtle *model; and (d)* uniform.

climbing the hierarchy is slower than descending the hierarchy when there is no locality and improves with locality. Finally, Keep, Jump, & Climb combines all the advantages and appears as the best solution in practice for this experiment: For low sparsity, it performs twice faster than the Delaunay hierarchy; and for high sparsity, it is as good as the Delaunay hierarchy (sometime even considerably faster).

From Scenario II (Figure 14 and 15). With a single walker, the spatial coherence is very strong and we expect a very good result for the last-point strategy since it highly benefits from previous location without any overhead for maintaining any structure of any kind. We also expect better results for the Climb strategy in that situation. This is indeed what happens, but Keep, Jump, & Walk with few landmarks, and Keep, Jump, & Climb remain quite close. When the number of walkers increases, the performance of last-point catastrophically falls down. The other strategies remain quite good until the number of walkers becomes higher than the number of landmarks, Keep, Jump, & Walk being the more resilient, since it requires a huge amount of landmarks. When the number of walkers becomes higher than the number of landmarks, the performance of the strategies fall down, specially the performance of Keep, Jump, & Walk with less landmarks. Keep, Jump, & Walk, Keep, Jump, & Climb, and the Delaunay hierarchy remain comparable. Overall, Keep, Jump, & Climb is as good as the Delaunay hierarchy when there is no spatial coherence,

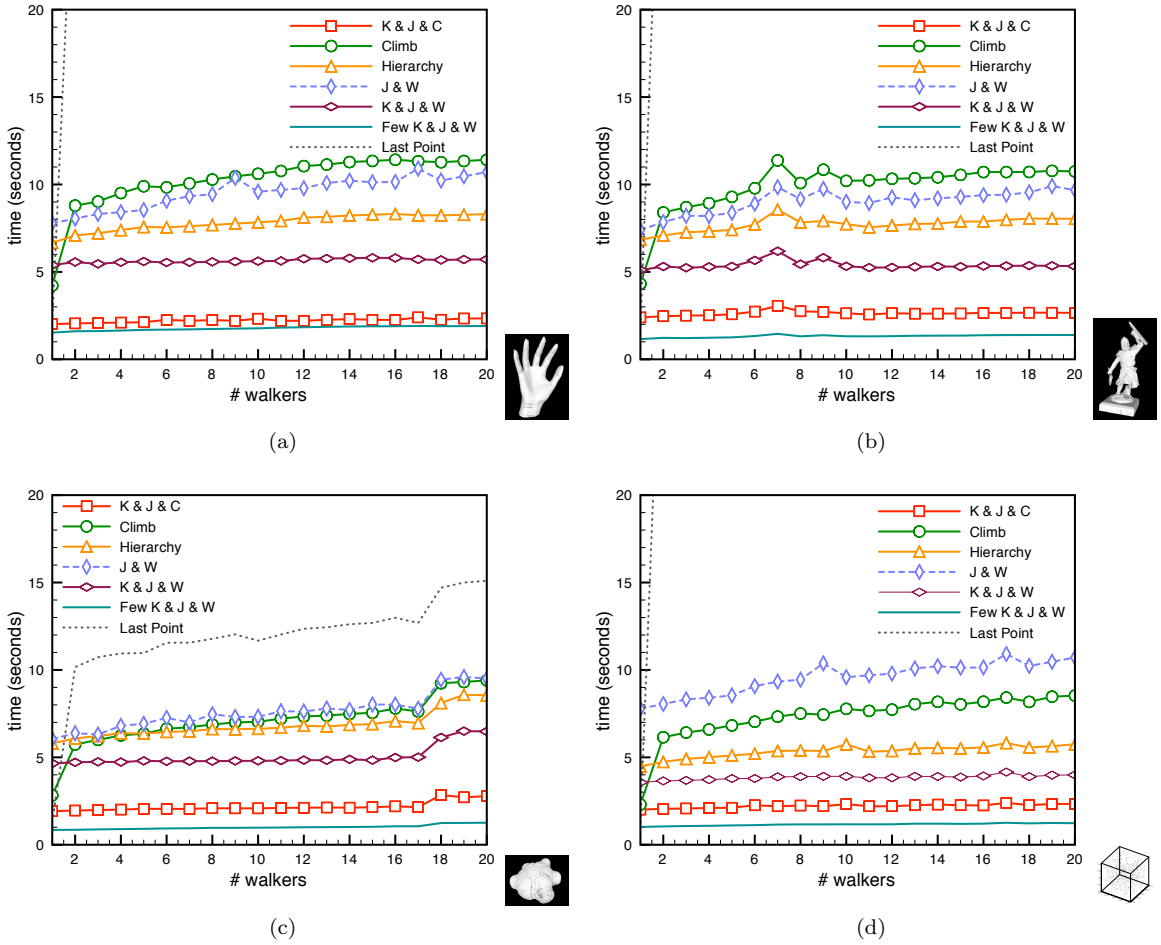


Figure 14: **Results for scenario II (coherence of queries).** *Computation times of the various algorithms in function of the number of parallel random walkers for: (a)* POORAN'S HAND *model; (b)* GALAAD *model; (c)* TOY TURTLE *model; and* UNIFORM. *Less random walkers mean strong spatial coherence (a single walker pushes that to an extreme).*

and a lot faster when there is spatial coherence. Moreover, Keep, Jump, & Climb is strictly better than Keep, Jump, & Walk in theory, and even better than the Delaunay hierarchy.

8 Conclusion

Our aim was to improve in practice the performance of point location in triangulation and we are mostly interested in

- queries with spatial coherence
- inside 3D triangulations
- in the CGAL library.

Before starting this work, our best data structure for this purpose was the Delaunay hierarchy, which was able to handle 1M queries in a 1M points triangulation in about 12 seconds for various scenarios. We have proposed Keep, Jump, & Climb, a new way of using the Delaunay hierarchy, which is not slower and often

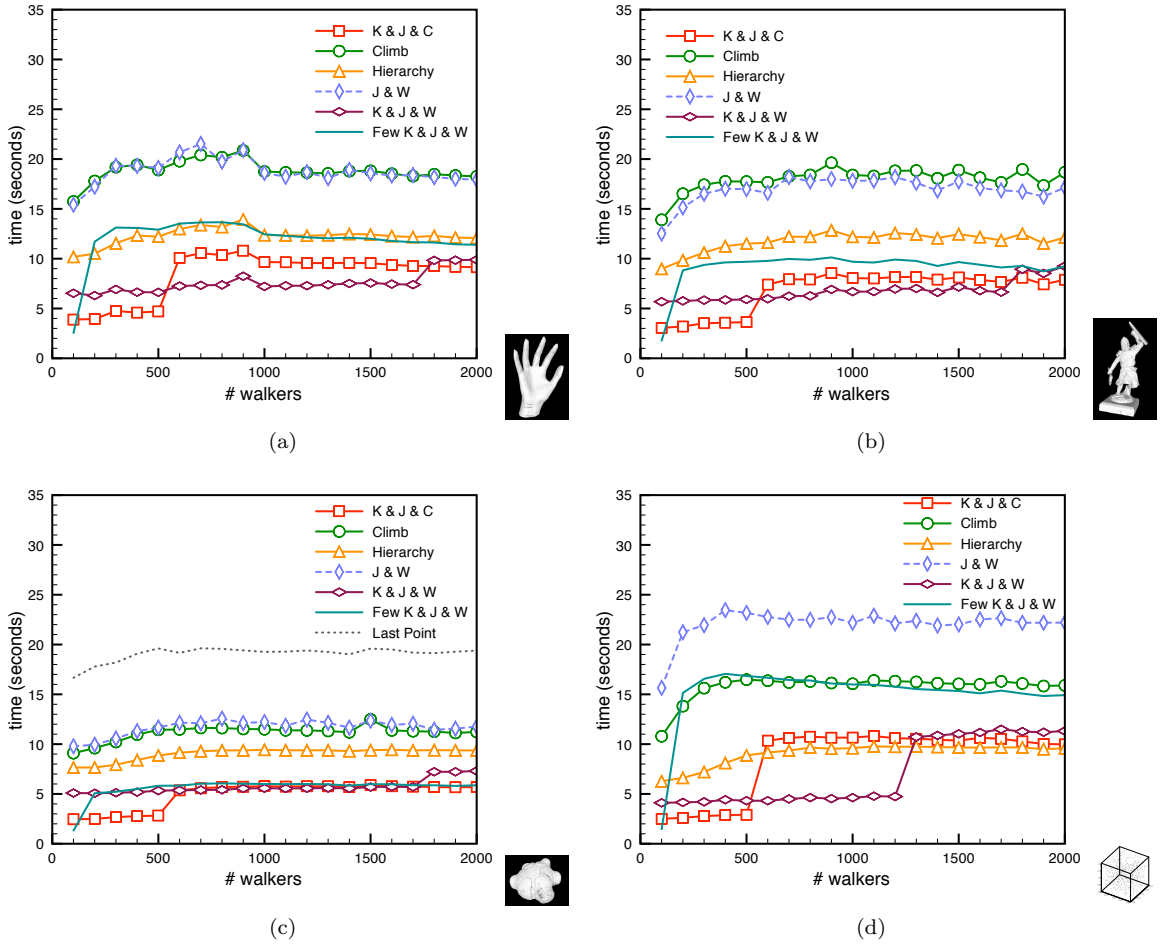


Figure 15: **Results for scenario II (number of walkers surpassing the number of landmarks).** ***The last-point strategy is sometime too slow to appear in the figures.***

significantly faster than the classical descent of the Delaunay hierarchy in our experiments. For a reasonable amount of spatial coherence of the queries, running time are improved by a factor 2. (In Figure 13, Figure 14, and, Figure 15 the running times vary from around 1s to around 11s depending on the query coherence; this should be contrasted with the classical Delaunay hierarchy, which achieves from around 5s to around 12s.)

One of our main tool in the theoretical analysis of our work is the introduction of the *Distribution Condition* that relates the expected number of tetrahedra intersected by a line segment with its length. It allows to analyze algorithms in a more general context than Delaunay triangulation of evenly distributed points. For example, we can derive from our work that the best size of sample for Keep, Jump, & Walk, when the data points lie on a surface, is $n^{3/8}$ and not the usual $n^{1/4}$. Furthermore, our experiments show that the Distribution Condition actually corresponds to some practical cases.

From a theoretical point of view, climbing the Delaunay hierarchy provides a solution to the problem of distribution-sensitive point location, which is much simpler and faster than previous data structures [31, 18, 33], but requires some reasonable hypotheses on the point set.

9 Open problems

The Distribution Condition brings several questions for the computational geometers. The first question is the one raised in Conjecture 2:

Question 15. *Do the Delaunay triangulations of n points evenly distributed on a bounded δ-dimensional manifold embedded in the d-dimensional space behave similarly to points evenly distributed in the Euclidean space of dimension δ with respect to the Distribution Condition?*

If Conjecture 2 has an affirmative answer, then walking on such triangulations does not depend on the ambient dimension, but only on the manifold dimension.

Figure 8 and 9 invite us to believe in a positive answer even if the points are not actually evenly distributed (they come from a laser scan), thus we may wonder what are actually the hypotheses needed by the conjecture.

Question 16. *What hypotheses a sampling of a bounded δ-dimensional manifold embedded in the d-dimensional space should verify such that the Delaunay triangulation satisfies the Distribution Condition with $\mathcal{F}(n) = n^{1/\delta}$?*

As a final remark, we want to insist on the dichotomy between the straight and stochastic walk. The straight walk is used in theoretical analysis for simplicity and usage of previous results, while the visibility walk is used in practice since it is faster and easier to implement. Thus an interesting research direction is to get a better theoretical basis for the use of the stochastic walk. This leads us to our last question:

Question 17. *Does Conjecture 1 have an affirmative answer?*

Acknowledgments The authors wish to thank FACEPE/INRIA, APQ-0055-1.03/12, for financial support, and Aim@shape for providing the realistic models.

References

[1] Core number library. http://cs.nyu.edu/exact/core_pages.

[2] The Gnu multiple precision arithmetic library. http://gmplib.org/.

[3] Leda, Library for efficient data types and algorithms. http://www.algorithmic-solutions.com/leda.

[4] N. Amenta, D. Attali, and O. Devillers. Complexity of Delaunay triangulation for points on lower-dimensional polyhedra. In *Proc. 18th ACM-SIAM Sympos. Discrete Algorithms*, pages 1106–1113, 2007.

[5] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Trans. Algorithms*, 3(2):17, 2007.

[6] D. Attali and J.-Daniel Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete Comput. Geom.*, 31:369–384, 2004.

[7] D. Attali, J.-D. Boissonnat, and André Lieutier. Complexity of the Delaunay triangulation of points on surfaces: The smooth case. In *Proc. 19th Annu. Symp. Comp. Geom.*, pages 237–246, 2003.

[8] J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Math. Proc. Camb. Phil. Soc.*, 55:299–327, 1959.

[9] P. Bose and L. Devroye. On the stabbing number of a random Delaunay triangulation. *Comput. Geom. Theory Appl.*, 36(2):89–105, 2007.

[10] A. Bowyer. Computing Dirichlet tesselations. *Comput. J.*, 24:162–166, 1981.

[11] CGAL Editorial Board. *CGAL User and Reference Manual*, 3.6 edition, 2010. `http://www.cgal.org`.

[12] J.-L D. Carufel, C. Dillabaugh and A. Mahesgwari. Point location in well-shaped meshes using jump-and-walk. *Proc. CCCG'11*, 147–152, 2011.

[13] P. M. M. de Castro and O Devillers. On the asymptotic growth rate of some spanning trees embedded in $\mathbb{R}^d$. Operations Research Letters, 39(1):44–48, 2011.

[14] P. M. M. de Castro and O Devillers. A pedagogic JavaScript program for point location strategies. Demo, in *Proc. 27th Annu. Symp. Comp. Geom.*, pages 295–296, 2011. `http://www.computational-geometry.org/SoCG-videos/socg11video/`

[15] B. Chazelle, D. Liu, A. Magen. Sublinear Geometric Algorithms. SIAM J. Comput., 35:627–646, 2006.

[16] C. Delage. Spatial sorting. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[17] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The Open Problems Project. http://www.cs.smith.edu/ orourke/TOPP/.

[18] Erik D. Demaine, John Iacono, and Stefan Langerman. Proximate point searching. *Comput. Geom. Theory Appl.*, 28(1):29–40, 2004.

[19] O. Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002.

[20] O. Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *Internat. J. Found. Comput. Sci.*, 13:181–199, 2002.

[21] L. Devroye, C. Lemaire, and J.-M. Moreau. Expected time analysis for Delaunay point location. *Comput. Geom. Theory Appl.*, 29:61–89, 2004.

[22] L. Devroye, E. P. Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. *Algorithmica*, 22:477–482, 1998.

[23] R. Dwyer. Convex hulls of samples from spherically symmetric distributions. *Discrete Appl. Math.*, 31(2):113–132, 1991.

[24] S. Funke, K. Mehlhorn, and S. Näher. Structural filtering: A paradigm for efficient and exact geometric programs. In *Proc. 11th Canadian Conf. on Comp. Geom.*, 1999.

[25] J. Gao and J. M. Steele. General spacefilling curve heuristics and limit theory for the traveling salesman problem. *J. Complexity*, 10:230–245, 1994.

[26] M. J. Golin and H.-S. Na. On the average complexity of 3d- Voronoi diagrams of random points on convex polytopes. *Comput. Geom. Theory Appl.*, 25:197–231, 2003.

[27] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, 2004. 2nd edition.

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

[29] I. Haran and D. Halperin. An experimental study of point location in planar arrangements in CGAL. *J. Exp. Algorithmics*, 13:2.3–2.32, 2009.

[30] J. Iacono. Improved upper bounds for pairing heaps. In *Proc. 7th Scandinavian Workshop on Algorithm Theor.*, pages 32–45, London, UK, 2000. Springer-Verlag.

[31] J. Iacono and S. Langerman. Proximate planar point location. In *Proc. 19th Annu. Symp. Comp. Geom.*, pages 220–226, 2003.

[32] J. Iacono. Expected asymptotically optimal planar point location. *Comput. Geom. Theory Appl.*, 29(1):19–22, 2004.

[33] J. Iacono. A static optimality transformation with applications to planar point location. In *Proc. 27th Annu. Symp. Comp. Geom.*, pages 21–26, 2011.

[34] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. 4th Eurographics Symp. on Geom. Processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[35] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. In *Proc. 12th European Symp. on Algorithms*, volume 3221 of *Lecture Notes Comput. Sci.*, pages 702–713. Springer-Verlag, 2004.

[36] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[37] D. E. Knuth. Optimum binary search trees. *Acta Inf.*, 1:14–25, 1971.

[38] T. Malamatos. Lower bounds for expected-case planar point location. *Comput. Geom. Theory Appl.*, 39(2):91–103, 2008.

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

[40] S. Pion and M. Teillaud. 3D triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010. `http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Triangulation_3/Chapter_main.html`

[41] L. K. Platzman and J. J. Bartholdi, III. Spacefilling curves and the planar travelling salesman problem. *J. ACM*, 36(4):719–737, October 1989.

[42] L. A. Santaló. *Integral Geometry and Geometric Probability*. Addison-Wesley, 1976.

[43] J. R. Shewchuk. Stabbing Delaunay tetrahedralizations. *Discrete Comput. Geom*, 32:343, 2002.

[44] Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2008.

[45] J. M. Steele. Cost of sequential connection for points in space. *Oper. Res. Lett.*, 8:137–142, 1989.

[46] J. M. Steele and T. L. Snyder. Worst-case growth rates of some classical problems of combinatorial optimization. *SIAM J. Comput.*, 18:278–287, 1989.

[47] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.

[48] M. Yvinec. 2D triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010. `http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Triangulation_2/Chapter_main.html`

[49] B. Zhu. Adaptive point location with almost no preprocessing in Delaunay triangulations. In *Proc. ISVD'12*, pages 84–89, 2012.