Practical Ways to Accelerate Delaunay Triangulations

Pedro M. M. de Castro

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention: Informatique

Présentée par

Pedro Machado Manhães de Castro

Méthodes pour Accélérer
les Triangulations de Delaunay

Thèse préparée dans le projet GEOMETRICA, INRIA Sophia-Antipolis

Dirigée par

Olivier Devillers, Directeur de Recherche, INRIA Sophia-Antipolis

Soutenue publiquement le 25 Octobre 2010 devant le jury composé de :

Président :	Bruno LÉVY, Directeur de Recherche, INRIA Loria
Rapporteurs :	Günter ROTE, Professeur, Freie Universität Berlin
	Franz AURENHAMMER, Professeur, Technische Universitaet Graz
Examinateurs :	Dominique ATTALI, Chargée de Recherche, CNRS
	Raphaëlle CHAINE, Maître de Conférences, LIRIS
	Olivier DEVILLERS, Directeur de Recherche, INRIA Sophia-Antipolis

MACHADO MANHÃES DE CASTRO Pedro

MÉTHODES POUR ACCÉLÉRER

LES TRIANGULATIONS DE DELAUNAY

Résumé: Méthodes pour Accélérer les Triangulations de Delaunay

Cette thèse propose de nouvelles méthodes pour accélérer certaines des plus importantes opérations dans une triangulation de Delaunay, conciliant efficacité et bonne complexité théorique.

Nous proposons deux approches pour calculer la triangulation de Delaunay de points sur (ou proches) d'une sphère. La première approche calcule la triangulation de Delaunay de points exactement sur la sphère par construction. La deuxième approche calcule directement l'enveloppe convexe de l'ensemble d'entrée, et donne quelques garanties sur la sortie. Les deux approches sont basées sur la triangulation régulière sur la sphère. La deuxième approche améliore les solutions de l'état de l'art.

L'operation de mise à jour d'une triangulation de Delaunay, quand les sommets bougent, est critique dans plusieurs domaines d'applications. Quand tous les sommets bougent, reconstruire toute la triangulation est étonnamment une bonne solution en pratique. Toutefois, lorsque les points se déplacent tres peu, ou si seulement une fraction des sommets bougent, la reconstruction n'est plus la meilleure option. Nous proposons un système de filtrage basé sur le concept de tolérance d'un sommet. Nous avons mené plusieurs expériences pour évaluer le comportement de l'algorithme sur des ensembles de données variés. Les expériences ont montré que l'algorithme est particulièrement pertinent pour les régimes convergents tels que les itérations de Lloyd. En dimension deux, l'algorithme présenté est un ordre de grandeur plus rapide que la reconstruction pour les itérations de Lloyd. En dimension trois, l'algorithme présenté a des performances équivalentes à la reconstruction quand tous les sommets bougent, cependant il est entièrement dynamique et améliore les solutions dynamiques précédentes. Ce résultat permet d'aller plus loin dans le nombre d'itérations de façon à produire des maillages de qualité supérieure.

La localisation de points dans une subdivision de l'espace est un classique de la géométrie algorithmique; nous réexaminons ce problème dans le cas des triangulations de $\mathbb{R}^d$ pour exploiter une éventuelle cohérence entre les requêtes. Nous analysons, implementons, et évaluons une strategie de localisation de point adaptable aux distributions des requêtes, basée sur Jump & Walk, appellée Keep, Jump, & Walk. Pour des paquets de requêtes, l'idée principale est d'utiliser les requêtes précédentes pour améliorer le traitement de la requête courante. Maintenant à propos de la complexité d'une requête dans une triangulation de Delaunay, nous montrons que la hiérarchie de Delaunay peut être utilisée pour localiser un point q à partir d'une requête précédente p avec une complexité randomisée $O(\log \sharp(pq))$ pourvu que la triangulation vérifie certaines hypothèses ($\sharp(s)$ désigne le nombre de simplex traversés par le segment s). Finalement, nous combinons la bonne adaptabilité à la distribution des requêtes du Keep, Jump, & Walk, et la bonne complexité de la hiérarchie de Delaunay, en une nouvelle stratégie de localisation de points appellée Keep, Jump, & Climb. Selon nos connaissances, Keep, Jump, & Climb est le premier algorithme adaptable aux distributions des requêtes qui marche en pratique et en théorie pour les triangulations de Delaunay—dans nos expérimentations, Keep, Jump, & Climb est plus rapide que la hiérarchie de Delaunay indépendamment de la cohérence spatiale des requêtes, et significativement plus rapide quand la cohérence spatiale est forte.

MACHADO MANHÃES DE CASTRO Pedro

PRACTICAL WAYS TO ACCELERATE

DELAUNAY TRIANGULATIONS

Abstract: Practical Ways to Accelerate Delaunay Triangulations

This thesis proposes several new practical ways to speed-up some of the most important operations in a Delaunay triangulation.

We propose two approaches to compute a Delaunay triangulation for points on or close to a sphere. The first approach computes the Delaunay triangulation of points placed exactly on the sphere. The second approach directly computes the convex hull of the input set, and gives some guarantees on the output. Both approaches are based on the regular triangulation on the sphere. The second approach outperforms previous solutions.

Updating a Delaunay triangulation when its vertices move is a bottleneck in several domains of application. Rebuilding the whole triangulation from scratch is surprisingly a viable option compared to relocating the vertices. However, when all points move with a small magnitude, or when only a fraction of the vertices moves, rebuilding is no longer the best option. We propose a filtering scheme based upon the concept of vertex tolerances. We conducted several experiments to showcase the behavior of the algorithm for a variety of data sets. The experiments showed that the algorithm is particularly relevant for convergent schemes such as the Lloyd iterations. In two dimensions, the algorithm presented performs up to an order of magnitude faster than rebuilding for Lloyd iterations. In three dimensions, although rebuilding the whole triangulation at each time stamp when all vertices move can be as fast as our algorithm, our solution is fully dynamic and outperforms previous dynamic solutions. This result makes it possible to go further on the number of iterations so as to produce higher quality meshes.

Point location in spatial subdivision is one of the most studied problems in computational geometry. In the case of triangulations of $\mathbb{R}^d$, we revisit the problem to exploit a possible coherence between the query points. We 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 retrieval of the current one. 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 $O(\log \sharp(pq))$ randomized expected complexity, where p is a previously located query and $\sharp(s)$ indicates the number of simplices crossed by the line segment s. 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—in our experiments, it is faster than the Delaunay hierarchy regardless of the spatial coherence of queries, and significantly faster when queries have reasonable spatial coherence.

This thesis is

dedicated to
the memory of
my grandma
Eunice Manhães de Castro

Publications

Some articles I have co-written during my thesis.

Articles in Journal or Book Chapters

- P. M. M. de Castro and O. Devillers. Practical Self-Adapting Point Location, in Triangulations. Submitted, 2010.

- P. M. M. de Castro and O. Devillers. On the Asymptotic Growth Rate of Some Spanning Trees Embedded in $\mathbb{R}^d$. Submitted (accepted), 2010.

- P. M. M. de Castro and O. Devillers. Design of the CGAL 3D Spherical Kernel and Application to Arrangements of Circles on a Sphere. *Computational Geometry: Theory and Applications*, 42(6-7):536-550, 2009.

- P. M. M. de Castro, J. Tournois, P. Alliez and O. Devillers. Filtering relocations on a Delaunay triangulation. In *Computer Graphics Forum, 28*, pages 1465–1474, 2009. Note: Special issue for EUROGRAPHICS Symposium on Geometry Processing.

Articles in Conference

- P. M. M. de Castro and O. Devillers. Simple and Efficient Distribution-Sensitive Point Location, in Triangulations. Submitted, 2010.

- M. Caroli, P. M. M. de Castro, S. Loriot, O. Rouiller, M. Teillaud and C. Wormser. Robust and Efficient Delaunay Triangulations of Points on Or Close to a Sphere. In SEA '10:*Proceedings 9th International Symposium on Experimental Algorithms*, pages 462–473, 2010.

- P. M. M. de Castro and O. Devillers. Fast Delaunay Triangulation for Converging Point Relocation Sequences. In *Abstracts 25th. European Workshop on Computational Geometry*, pages 231-234, 2009.

Reports

- P. M. M. de Castro and O. Devillers. Walking Faster in a Triangulation. Research Report 7322, INRIA, 2010.

- P. M. M. de Castro and O. Devillers. On the Size of Some Trees Embedded in $\mathbb{R}^d$. Research Report 7179, INRIA, 2010.

- P. M. M. de Castro and O. Devillers. Self-Adapting Point Location. Research Report 7132, INRIA, 2010.

- M. Caroli, P. M. M. de Castro, S. Loriot, O. Rouiller, M. Teillaud and C. Wormser. Robust and Efficient Delaunay Triangulations of Points on Or Close to a Sphere. Research Report 7004, INRIA, 2009.

- P. M. M. de Castro and O. Devillers. Delaunay Triangulations for Moving Points. Research Report 6750, INRIA, 2008.

- P. M. M. de Castro and O. Devillers. State of the Art: Updating Delaunay Triangulations for Moving Points. Research Report 6665, INRIA, 2008.

- P. M. M. de Castro and M. Teillaud. CGAL 3D Spherical Kernel. Technical Report ACS-TR-363605-02, INRIA, 2008.

Acknowledgements

I owe my sincere gratitude to:

- **Olivier Devillers**, my **great** advisor.

  > *"To be ignorant of one's own ignorance is to be in an unprogressive, uninspired state of existence."* — *David O. McKay.*

  I cannot agree more; and I was lucky enough to strike several times my own ignorance along these three years of research, making my state of existence neither unprogressive nor uninspired. Thank you so much Olivier. I hope one day I will be for my students the great advisor, human being, and teacher you are for me.

- **Monique Teillaud**. Once in 2006, during my Master thesis in Recife, I saw a list of some internship proposals from the Institut National de Recherche en Informatique et Automatique (INRIA). Amongst the proposals, one took my attention more than the others: "Curved Kernel for CGAL", by Monique Teillaud. I decided to apply, and fortunately Monique accepted the application; four years later, Computational Geometry is one of my passions. I thank Monique for various reasons: (i) for introducing me to Computational Geometry; (ii) for introducing me to the GEOMETRICA team; (iii) for giving me the opportunity to work in the Computational Geometry Algorithms Library (CGAL); (iv) for making me part of the Circular Kernel 2D, Spherical Kernel 3D, and the "triangulation on sphere" projects (which is the subject of the Chapter 3 of my thesis); (v) for being so inspiring, kind, and supportive. (This list is far from exhaustive.) Thank you so much Monique.

- **Pierre Alliez, Manuel Caroli, Frédéric Cazals, Olivier Devillers, Sébastien Loriot, Sylvain Pion, Olivier Rouiller, Monique Teillaud, Jane Tournois, Camille Wormser.** I wish to thank them for the work we have done together: discussions, implementations, readings, and more . . .

- **The whole GEOMETRICA team.** Working in such a great research group was honorable, insightful, pleasant, and rewarding. I hope to spread several good practices I have learnt within GEOMETRICA during my career.

- **Dominique Attali, Franz Aurenhammer, Raphaëlle Chaine, Olivier Devillers, Bruno Lévy, and Günter Rote.** I wish to warmly thank them for accepting to be part of my thesis committee.

- **ANR-Triangles, and Région PACA.** These entities financially supported my work.

- **Centro de Informática da Universidade Federal de Pernambuco (CIn).** I owe gratitude to several nice teachers and researchers from CIn for several reasons: for showing to me in practice what the job of a researcher is, for introducing me to the ACM programming competitions, for passionate discussions on analytical geometry, on computational music, just to name a few ...

- **Hua Minh-Duc.** Sometimes we unexpectedly cross paths with someone transcendental in life. Breaking my foot while living at CIV is perhaps one of my greatest fortune in these years :). (Thank you so much my friend, I'm sure in no-time I'll be seeing your technological devices under water.)

- **Family and friends.** Paths and walls don't matter, we are together in this adventure. Obrigado! Merci! Thank you! Dank! Kiitos! ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pafnuty Lvovich Chebyshev, Andrey Andreyevich Markov, Georgy Feodosevich Voronoi, and Boris Nikolaevich Delaunay have all substantially contributed to the progress of science; their portraits are shown in Figure 1.1. Delaunay was a PhD student of Voronoi, Voronoi was a PhD student of Markov, who in turn was a PhD student of Chebyshev. They contributed in the field of probability, statistics, and number theory, to name a few; some of their contributions, such as *Chebyshev's inequality*, *Markov chains*, *Voronoi diagrams*[1], and **Delaunay triangulations** carry their signature.

| (a) Chebyshev | (b) Markov | (c) Voronoi | (d) Delaunay |

Figure 1.1: **Some memorable russian mathematicians.**

---

[1]Some researches manipulated these kind of diagrams before Voronoi. Informal use of *Voronoi diagrams* can be traced back to Descartes in 1644, and formal use can be traced back to Dirichlet in 1850; sometimes Voronoi diagrams are also called *Dirichlet tessellations*.

(a) Voronoi diagram   (b) Delaunay triangulation
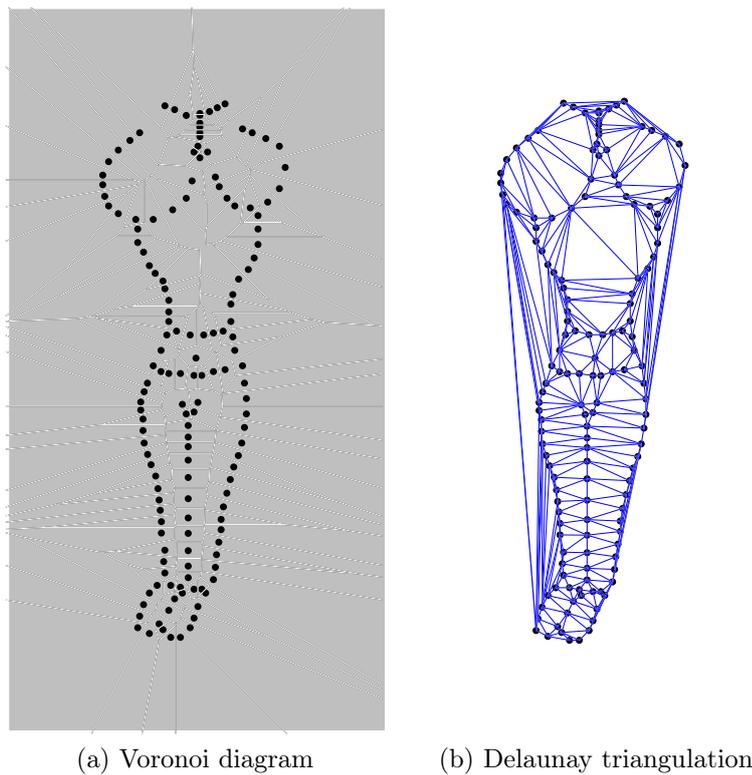
Figure 1.2: **Voronoi diagram and Delaunay triangulation.** *The space is $\mathbb{R}^2$, the metric is the Euclidean metric.*

The contributions of these mathematicians partially form the basis of several contemporary researches. In particular, *Voronoi diagrams* [219] and *Delaunay triangulations* [89] are extensively adopted in several fields of research, ranging from archeology to zoology [104].

Given a set of n points $S = p_1, p_2, \ldots, p_n$ in a space $\mathcal{Z}$, consider n regions $\mathcal{R}_i$ such that $\mathcal{R}_i$ contains all the points in $\mathcal{Z}$ *closer* to p_i than any other point p_j, $p_i \neq p_j$. The union of these regions is the Voronoi diagram of S; see Figure 1.2(a). The word *closer* in the definition above deserves a special treatment: A point p_i is *closer* to p_j compared to p_k if and only if $dist(p_i, p_j) < dist(p_i, p_k)$, where $dist$ is a distance function defined by a given metric on $\mathcal{Z}$. The Delaunay triangulation is simply the dual of the Voronoi diagram; see Figure 1.2(b). In this thesis, when the metric is not mentioned, the Euclidean metric is meant.

Aurenhammer [37] and Okabe et al. [175] give a comprehensive (though not exhaustive) list of different kinds of Voronoi diagrams, applications, and properties. The literature on Voronoi diagram and Delaunay triangulations is vast, the following three books are recommended [82, 182, 51]. In this thesis, our main focus is Delaunay triangulations; some Delaunay triangulations are depicted in Figure 1.3.

Delaunay triangulations have some very useful properties; here are an important sampling of them:

Property 1 ([57, 114]). *Each edge of the Delaunay triangulation of a set of points is a chord of at least one empty hypersphere; i.e., a hypersphere including no points in the set.*

Property 2 ([155, 205])**.** *In the plane, the Delaunay triangulation maximizes the minimum angle.*

Property 3 ([175, 92])**.** *The minimum spanning tree of S is a subgraph of the Delaunay triangulation.*

Property 4 ([175])**.** *The nearest neighbor graph of S is a subgraph of the Delaunay triangulation.*

Property 5 ([81, 185, 68])**.** *The Delaunay triangulation minimizes the roughness measure of a piecewise linear interpolation for a given function.*

We can find Delaunay triangulations on: data analysis [206], data clustering [136, 30], geographic databases [173], geophysics [118], information visualization [154], mesh generation [111, 87], mesh optimization [24, 216, 67], mesh smoothing [25], motion planning [192], re-meshing [218, 23], surface reconstruction [63, 146], to name a few. More applications (and details) can be found e.g., in the following texts [29, 37, 175, 209, 110]. It is not surprising that Delaunay triangulations found their place in several applications, the properties above show a few reasons to use them.

In some applications, such as data analysis, data clustering, and meshing, the size of the input sets increase with the available technology; consequently, the interest in practical and fast ways for constructing Delaunay triangulations keeps rising. Past researches led to four main ways to compute Delaunay triangulations: *sweep line* [119, 208, 202], *duality* [57, 114, 207], *divide-and-conquer* [197, 156, 132, 73], and *incremental* [156, 132, 199, 130, 93, 47]. Since the Delaunay triangulation is the dual of the Voronoi diagram, computing Delaunay triangulations and Voronoi diagrams is equivalent, and descriptions in what follows account for both.

Sweep line [119, 208, 202]. Fortune's sweep line algorithm is a classical algorithm to build Delaunay triangulations for points in the plane. The algorithm maintains a line that sweeps the plane from the leftmost point to the rightmost point of the input set. Each time the line passes through a point, the point is included in the Voronoi diagram. Fortune's algorithm computes Delaunay triangulations in $O(n \log n)$.

Duality [57, 193, 114, 194, 66, 207]. Delaunay triangulations in dimension d can be obtained from convex hulls in dimension $d + 1$ thanks to the duality between them. The procedure consists in: (i) lifting points in S onto the unit paraboloid one dimension higher; (ii) computing the lower hull of the lifted points; then (iii) projecting back the lower hull onto $\mathbb{R}^d$. This approach delegates the computation of a Delaunay triangulation to the computation of a lower hull one dimension higher. The lower hull of a set of points in dimension d can be computed in $O(n \log n + n^{\lfloor d/2 \rfloor})$ for any fixed dimension; i.e., using duality, Delaunay triangulations in dimension d can be computed in worst-case optimal $O(n \log n + n^{\lceil d/2 \rceil})$.

Divide-and-conquer [197, 156, 132, 73]. The divide-and-conquer algorithms divide the problem of computing the Delaunay triangulation of a point set in: (i) computing the Delaunay of two disjoint subsets of the input set, and (ii) merging the solutions of the two smaller problems into the solution of the original problem; this process is then applied recursively. In two dimensions, the divide-and-conquer algorithms can typically compute Delaunay triangulations in $O(n \log n)$ time.

Incremental [156, 132, 199, 130, 47, 93]. The incremental algorithms start with an *empty* Delaunay triangulation, then inserts one point after another, until all the points

of the input set are inserted. The result is the Delaunay triangulation of the whole input set. Incremental construction of the Delaunay triangulation can be held in a worst case $O\left(n^{\lceil d/2\rceil+1}\right)$ complexity; however, in randomized analyses, the incremental construction has a $O(n\log n + n^{\lceil d/2\rceil})$ complexity. (Under some hypotheses on the size of the output $O(n\log n)$ suffices for any finite dimension [93].)

The incremental algorithms gained special attention of the scientific community, for several reasons: (i) the point set does not need to be known in advance; (ii) they work well in pair with randomization [76, 75]; (iii) they are often simpler to implement; and (iv) some of them behave well in practice also for dimensions higher than two. When points are known in advance, sorting the points can dramatically improve the performance of the incremental algorithm [27, 60, 88]. A very efficient implementation of the incremental Delaunay triangulation construction can be found in the Computational Geometry Algorithms Library (CGAL) [64].

Applications such as mesh optimization, mesh smoothing, motion planning, and re-meshing, require the Delaunay triangulation to be *dynamic*. A dynamic data structure is a data structure that can be maintained under **insertions** and **deletions** of points. Naive **relocations** of points are automatically supported by successively inserting and deleting a point. Efficient incremental algorithms support efficient dynamic insertions, and with some extra effort, efficient deletions [70, 94, 99] as well. The three operations cited above are amongst the most important operations on a Delaunay triangulation; however, the most fundamental[2] operation is **point location**: Given a query point, the point location retrieves the triangle of the Delaunay triangulation in which the query point lies in. Two- and three-dimensional Delaunay triangulations in CGAL support efficient point location [93], insertion [88], deletion [94, 99], and relocation.[3]

To sum up, Delaunay triangulations date back from 1934 [89] and is one of the most famous and successful data structures introduced in the field of Computational Geometry. Two main reasons explain this success: (i) it is suitable to many practical uses; and (ii) computational geometers have produced efficient implementations [223, 179, 201].

While implementations nowadays are very efficient, there is still room for improvements. And improvements are highly necessary for a broad range of applications. Interesting instances of applications such as geographic database [173], mesh optimization [217], and Poisson surface reconstruction [146] require several hundred millions of operations on Delaunay triangulations of some few millions of points; and this keeps increasing. In this thesis, we focus on designing algorithms to perform operations on a Delaunay triangulation even faster.

Contributions

This thesis proposes several new practical ways to speed-up some of the most important operations on a Delaunay triangulation. Our main focus is to reduce the computation time needed to perform operations in large input sets. We contribute with solutions for the following three problems:

---

[2]Insertions, deletions, and relocations depend on point location.

[3]Support for relocations is a new feature in CGAL 3.7, implemented during this thesis.

Problem 6 (Delaunay triangulations on the sphere). *"Given points on or close to the sphere, propose solutions to construct the Delaunay triangulation on the sphere of these points."*

We propose two efficient and robust approaches to compute a Delaunay triangulation for points on or close to a sphere. The first approach computes the Delaunay triangulation of points placed exactly on the sphere. The second approach computes the *convex hull* of the input set, and gives some guarantees on the output. The second approach outperforms previous solutions.

Problem 7 (Relocations on a Delaunay triangulation). *"Given a set of moving points, compute its Delaunay triangulation at some given discrete time stamps."*

Updating a Delaunay triangulation when its vertices move is a bottleneck in several applications. Rebuilding the whole triangulation from scratch is surprisingly a viable option compared to relocating the vertices. However, when all points move with a small magnitude, or when only a fraction of the vertices moves, rebuilding is no longer the best option. We propose a filtering scheme based upon the concept of vertex tolerances. We conducted several experiments to showcase the behavior of the algorithm for a variety of data sets. The experiments showed that the algorithm is particularly relevant for convergent schemes such as the *Lloyd iterations*. In two dimensions, the algorithm presented performs up to an order of magnitude faster than rebuilding for *Lloyd iterations*. In three dimensions, although rebuilding the whole triangulation at each time stamp when all vertices move can be as fast as our algorithm, our solution is fully dynamic and outperforms previous dynamic solutions. This result makes it possible to go further on the number of iterations so as to produce higher quality meshes.

Problem 8 (Point location in triangulations). *"Given a triangulation $\mathcal{T}$ and a set of query points, for each query point, find the cell in $\mathcal{T}$ containing it."*

Point location in triangulations of $\mathbb{R}^d$ is one of the most studied problems in computational geometry. We revisit the problem to exploit a possible coherence between the query points. We 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. 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 $O(\log \sharp(pq))$ randomized expected complexity, where p is a previously located query and $\sharp(s)$ indicates the number of *simplices* crossed by the line segment s. 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—in our experiments, it is faster than the *Delaunay hierarchy* regardless of the spatial coherence of queries, and significantly faster when queries have reasonable spatial coherence.

Overview of the Thesis

We start with Chapter 2, which describes some necessary notions and definitions that are extensively used in what follows. Then the rest of the thesis is divided into three parts

plus a conclusion. Each part concerns a different problem, and consists of one or more chapters.

• **Part I, Delaunay triangulations on the sphere**: Chapter 3. This chapter presents two approaches to compute Delaunay triangulations for points on or close to the sphere.

• **Part II, Relocations on a Delaunay triangulation**: Chapters 4 and 5. Chapter 4 briefly describes the state of the art of point relocations in a triangulation, both in theory and in practice. Then Chapter 5 presents a new dynamic filtering algorithm, which avoids unnecessary insertions and deletions of vertices when relocating them.

• **Part III, Point location in triangulations**: Chapters 6, 7, and 8. Chapter 6 reviews old results and presents new results on some spanning trees embedded in $\mathbb{R}^d$; these results are used in Chapter 8. Chapter 7 describes a sampling of the most important point location algorithms both in theory and practice. Then in Chapter 8, we present several new distribution-sensitive point location algorithms.

Finally, we conclude the work with Chapter 9 in Part IV, where we present some perspectives and we collect some of the most important open problems spread along this work.

(a)   (b)

(c)   (d)
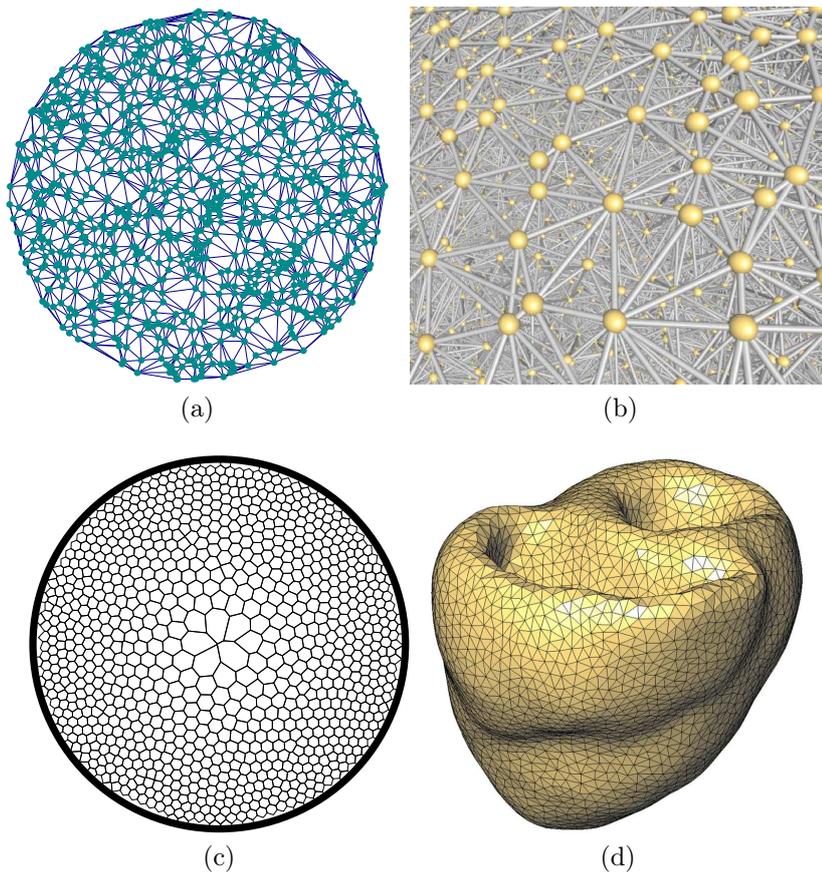
Figure 1.3: **Delaunay triangulation.** *(a) two-dimensional Delaunay triangulation of points in a disc; (b) three-dimensional Delaunay triangulation of points in a ball (picture taken from the CGAL [64]); (c) and (d) are useful artifacts that we can obtain using Delaunay triangulations (more details in Chapter 5).*

Chapter 2

Notions and Definitions

"All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value." — *Carl Sagan.*

This chapter presents the well-known *arithmetic filtering* technique, which has transformed a theoretical game called Computational Geometry in a practical tool. Then, some useful definitions about triangulations are given. Notions and definitions presented in this chapter are subsequently used in the rest of this thesis.

2.1 Exact Arithmetic

2.1.1 Predicate

In the whole generality, predicates are functions mappings elements from an arbitrary set of possible inputs to a finite set of *predicate results*. However, in computational geometry, it is often used to map geometrical primitives (e.g. points, lines, spheres) to signs (-1, 0, $+1$).

In this thesis, we assume an even less general setting: Predicates are the sign of some polynomial. Even if this setting is less abstract, it turns out that a great amount of useful predicates in computational geometry can be described as such.

Algorithms in computational geometry often rely on predicates. And these predicates must be computed exactly, otherwise the algorithm may even not terminate [147]. In other words, *robustness* of algorithms in computational geometry is related with the exact computation of predicates.

A way to compute predicates exactly is adopting the *exact geometric computation paradigm* from Yap and Dubé [221]. More precisely, computing signs with *exact number types* [6, 8, 7], which are multi-precision number types capable of representing exactly a rational (or more generally an algebraic number).

Exact number types are very slow. In the next section, we describe how to accelerate several orders of magnitude the exact predicate computations with a very practical *filtering* technic.

2.1.2   Accelerating Computations with Filtering

Exact predicate computation using naive exact number types is too slow. In order to avoid an excessive overhead, one could use a technic called *arithmetic filtering*.

Arithmetic filtering consists basically in: (i) computing an **approximate** value λ of the expression using fast *inexact number types*;[1] (ii) computing a bound Δ_λ on the maximum deviation between the approximate and **exact** value, called *error bound*, using inexact number types; (iii) checking whether $\lambda - \Delta_\lambda$ and $\lambda + \Delta_\lambda$ have the same sign, i.e., checking for a *filter success* (when the computation is certified); and finally, in the case of a *filter failure* (when the computation is not certified), (iv) computing the exact value using exact number type. Such a procedure is schematized in Figure 2.1.

There are three main derivations of the arithmetic filtering; their main difference lies in how inexact computations are certified:

Static Filtering. If the error bound can be computed at compile time, e.g., when the expression uses only $+, -, \times$, and an upper bound on the inputs is known, then the inexact computation can be certified with just one additional comparison. We call this filtering scheme a *static filtering*. The next two schemes are less restrictive.

Semi-Static Filtering. Some arithmetic operations, such as division and square root, require the knowledge of a lower bound on the inputs in order to upper-bound the error. This makes upper-bounding errors of an expression hard to handle statically because of the lower bounds on the intermediate results. A solution consists in: (i) computing an upper bound of the *relative error*[2] of the expression at compile time; then, for each input of the expression, (ii) computing its *order*[3] at compile time and (iii) computing its upper bound[4] at running time; and finally, (iv) multiplying the relative errors by the upper bound of each input to the power of its corresponding order. We call this filtering scheme a *semi-static filtering*. We can find implementations of semi-static filtering in CGAL [64]. However, sometimes the relative error bounds are too pessimistic.

Dynamic Filtering. In this scheme, for every arithmetic operation, error bounds for the results are computed on the fly; then, the results are compared against these error bounds. This can be done automatically using interval arithmetic [56]. However, as a drawback, dynamic filtering is at least twice slower than static filtering [98, 165].

The three categories of arithmetic filtering above can be combined in order to improve performances.[5] In the next section we show how the value of an expression computed inexactly can be bounded in the case of the (IEEE 754) double precision floating-point number type.

---

[1]e.g., `float` and `double` in C and C++.

[2]The relative error of an expression computation is the (absolute) error divided by the exact value.

[3]An order of an input is its power in the expression; e.g., an area has an order of 2 (meter squared).

[4]Sometimes also computing lower bounds if necessary; e.g. when the expressions use a division.

[5]CGAL combines semi-static and dynamic filtering for better performances.
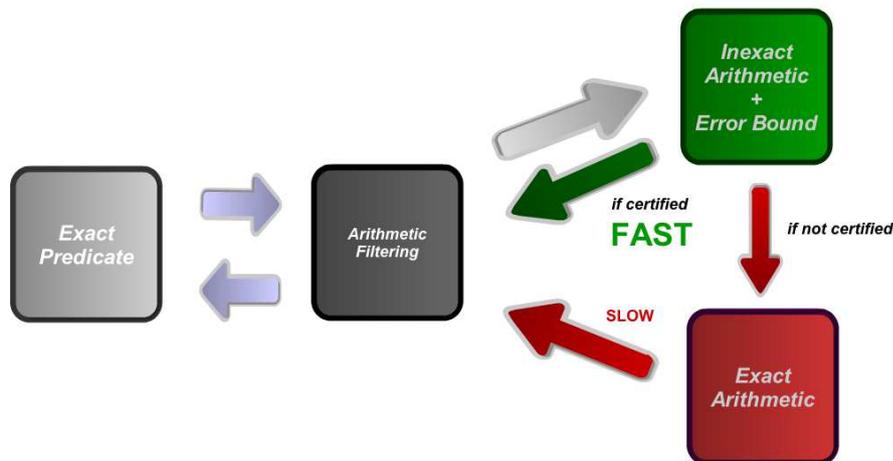
Figure 2.1: **Arithmetic filtering.**

2.1.3 Bounding Expressions Computed With Double

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely-used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point numbers and special values together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions. The default rounding mode is the *"Round to Nearest"*, which naturally consists of rounding to the nearest representable value; if the number fails midway, it is rounded to the nearest value with an even least significant bit. These rules for rounding apply to the four basic operations $(+, -, \times, /)$ and also to the square root operation. This rounding mode leads to *absolute errors* smaller than $\mathrm{ulp}(x)/2$, with $\mathrm{ulp}(x)$ being the *units in the last place* of x (for standard `double` it is 2^{-53}). For an easy-to-read survey on floating-point arithmetic fundamentals, please refer to [124].

Let x be the exact value of a given expression ξ, and $f(x)$ the value obtained by computing ξ with floating-point numbers; then next, we obtain α_x such that $|f(x) - x| \leq \alpha_x$, for any expression ξ composed by the following five operations: $(+, -, \times, /, \sqrt{})$. The value α_x is a *certified error bound* of x; i.e., a value such that the interval $[f(x) - \alpha_x, \ f(x) + \alpha_x]$ contains the exact value x.

If $f(a)$ and $f(b)$ are the resulting floating-point evaluations with their certified error bounds α_a and α_b respectively, then a certified error bound $\alpha_{a \pm b}$ of $a \pm b$ can be obtained as follows:

$$|f(a \pm b) - (a \pm b)| \leq \alpha_{a \pm b} = \alpha_a + \alpha_b + |a \pm b|\frac{\mathrm{ulp}(1)}{2}, \tag{2.1}$$

Besides, a certified error bound $\alpha_{a \times b}$ of $a \times b$ can be obtained as follows:

$$|f(a \times b) - (a \times b)| \leq \alpha_{a \times b} = \alpha_a |b| + \alpha_b |a| + |a||b|\frac{\mathrm{ulp}(1)}{2}. \tag{2.2}$$

For the division and the square root operation, we cannot bound their floating-point value because the inverse of an expression cannot be bounded statically. However, the inverse

of an expression can be bounded on the fly. Then, a certified error bound $\alpha_{1/a}$ of $1/a$, assuming $A = |f(a) - \alpha_a| \leq |a|$ and $\alpha_a < A \leq |f(a)|$, can be obtained as follows:

$$\left| f\left(\frac{1}{a}\right) - \left(\frac{1}{a}\right) \right| \leq \alpha_{1/a} = \frac{\alpha_a}{A^2} + \left|\frac{1}{A}\right| \frac{\text{ulp}(1)}{2}. \tag{2.3}$$

The equation above depends on A and α_a. If the above-mentioned condition $\alpha_a < A \leq |f(a)|$ does not hold, the error bound is no longer certified, and a filter failure should be signaled. Moreover, if $f(a)$ is a non-degenerated floating-point expression with $a \geq 0$, then a certified error bound $\alpha_{\sqrt{a}}$ of $\sqrt{a}$ can be obtained as follows:

$$\left| f(\sqrt{a}) - \sqrt{a} \right| \leq \alpha_{\sqrt{a}} = \frac{\alpha_a}{2\sqrt{A}} + \sqrt{a}\frac{\text{ulp}(1)}{2}. \tag{2.4}$$

Recall that $|f(a)| - \alpha_a \leq |a| \leq |f(a)| + \alpha_a$, and hence $|a|^{-1/2} \leq (|f(a)| - \alpha_a)^{-1/2}$.

An example on how such error computations are used in practice is shown in Section 5.3 with details.

2.2 Triangulations

2.2.1 Definitions

A *k-simplex* is a polytope of dimension k with $k + 1$ vertices; see Figures 2.2(a), 2.2(b), 2.2(c), and 2.2(d). A *face* of a k-simplex σ is a l-simplex contained in σ, with $l < k$. A simplicial complex is a set $\mathcal{T}$ of simplices such that: (i) any face of a simplex in $\mathcal{T}$ is also in $\mathcal{T}$, and (ii) the intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{T}$ is a face of both σ_1 and σ_2; see Figures 2.2(g) and 2.2(h). The dimension d of a simplicial complex is the maximum dimension of its simplices. A d-simplex, a $(d-1)$-simplex, and a $(d-2)$-simplex are also called a *cell*, *facet*, and a *ridge* respectively; see Figures 2.2(a), 2.2(b), 2.2(c).

A simplicial complex $\mathcal{T}$ is pure if any simplex of $\mathcal{T}$ is included in a cell of $\mathcal{T}$. Two cells in $\mathcal{T}$ are said to be adjacent if they share a facet. A simplicial complex is connected if the adjacency relation defines a connected graph over the set of cells of $\mathcal{T}$. The union $\mathcal{U}_\mathcal{T}$ of all simplices in $\mathcal{U}_\mathcal{T}$ is called the domain of $\mathcal{U}_\mathcal{T}$. A point p in the domain of $\mathcal{T}$ is said to be singular if its surrounding in $\mathcal{U}_\mathcal{T}$ is neither a topological ball nor a topological disc. A d-dimensional triangulation is a d-dimensional simplicial complex that is pure, connected, and without singularity; see Figures 2.2(e) and 2.2(f).

Let $\mathcal{T}$ be a triangulation and $\mathcal{Z}$ a given space, let each simplex of $\mathcal{T}$ be assigned a subset of $\mathcal{Z}$, then the set of all these subsets, one for each simplex, is the *embedding* of $\mathcal{T}$ in $\mathcal{Z}$; this definition however is too abstract for the scope of this thesis. Next, we instantiate a more specific definition of the embedding of $\mathcal{T}$, which is used throughout this work.

Let $\mathcal{T}$ be a triangulation. We define the graph (V, E) of $\mathcal{T}$, where V and E are the set of 0-simplices and 1-simplices of $\mathcal{T}$ respectively, as the *combinatorics* or *combinatorial structure* of $\mathcal{T}$. We call a vertex of the combinatorial structure of $\mathcal{T}$ a *vertex* of $\mathcal{T}$ for short. Let each vertex of $\mathcal{T}$ be assigned a point in $\mathbb{R}^d$, then the structure formed by linking points by segments of line according to the combinatorics of $\mathcal{T}$ is called the *embedding* of $\mathcal{T}$ in $\mathbb{R}^d$; or simply the embedding of $\mathcal{T}$, when there is no ambiguity. Note that a valid triangulation can have an embedding that is an invalid triangulation; this happens
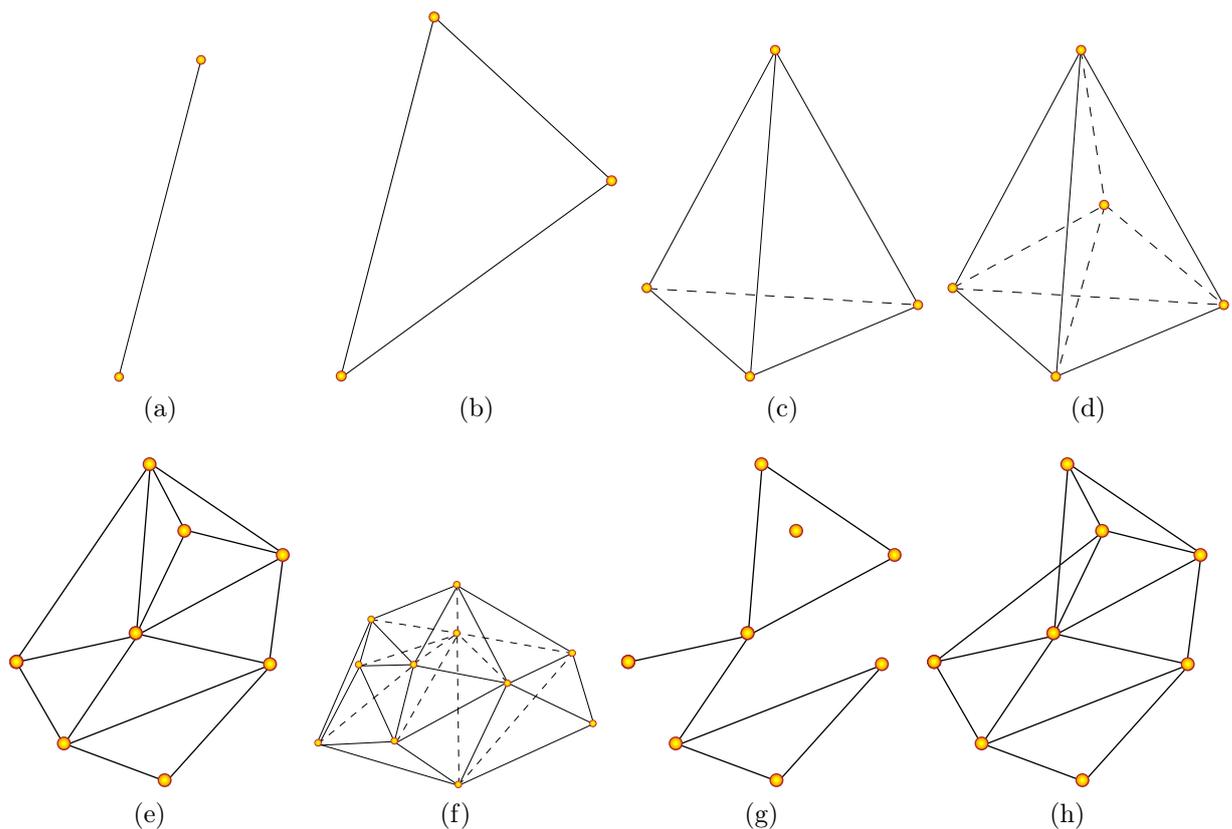
Figure 2.2: **Simplex, simplicial complex, and triangulation.** *(a) a 1-simplex (a three-dimensional ridge); (b) a 2-simplex (a three-dimensional facet); (c) a 3-simplex (a three-dimensional cell); (d) a 4-simplex; (e) a two-dimensional triangulation; (f) a three-dimensional triangulation; (g) a simplicial complex; (h) not a simplicial complex.*

e.g., when the embedding of $\mathcal{T}$ becomes self-intersecting, such as the example depicted in Figure 2.2(h): The valid triangulation shown in Figure 2.2(e) can be made invalid by switching the embedding of two vertices; see Figure 2.2(h).

2.2.2   Representation

There are several ways to represent triangulations as a data structure; see [182, 201, 223, 179] for a comprehensive list. In this thesis, we use CGAL's 2D and 3D triangulation representations [223, 179], which we succinctly describe next.

In CGAL's representation, only cells and vertices are actually stored in memory. Each cell σ has an array $\mathcal{A}_\sigma$ and an array $\mathcal{V}_\sigma$ of $d+1$ neighbor pointers and vertex pointers respectively: pointers in $\mathcal{A}_\sigma$ point to distinct adjacent cells; and pointers in $\mathcal{V}_\sigma$ point to distinct vertices of σ. Indexing $\mathcal{A}_\sigma$ and $\mathcal{V}_\sigma$ by i, $0 \le i \le d$, gives respectively: a cell σ_i, and the vertex opposite to σ_i. A vertex v has a coordinate p corresponding to its embedding, and a pointer to an incident cell; such pointer is necessary to efficiently navigate inside the triangulation. Figure 2.3 depicts the representation of a two-dimensional cell and its three vertices. Additional data can be easily annexed to cells and vertices. Finally, the set of cells and the set of vertices are stored in two separate containers. Such a representation
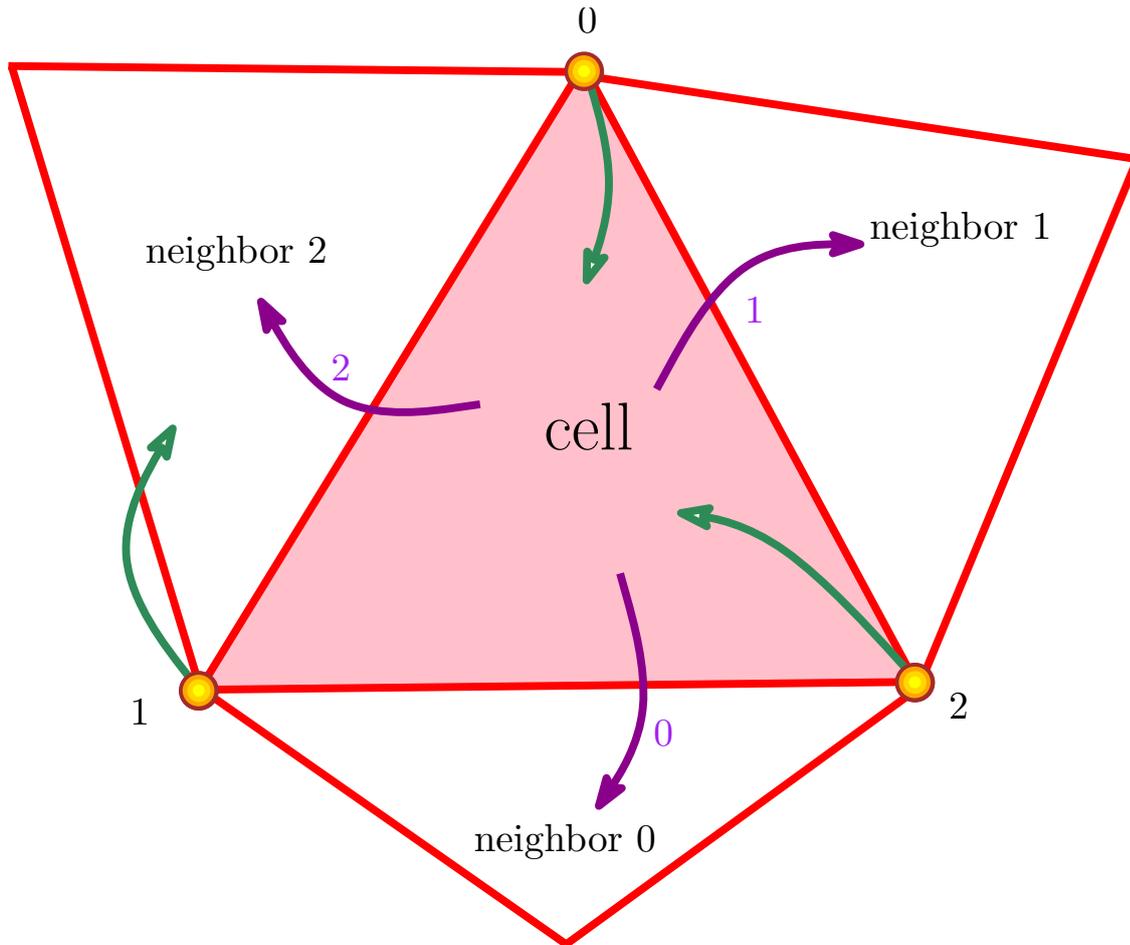
Figure 2.3: **Representation of a cell and its vertices.**

gives a constant-time access between adjacent cells, which is useful when performing point location; this feature is used in Section 7.3.1.

CGAL adopts the following common practice to easily handle the boundary of $\mathcal{T}$: It consists in adding a "point at infinity" ∞ to the initial set of points [20, 223, 179]; see Figure 2.4. Let S be the initial set of points, we consider an augmented set $S' = S \cup \{\infty\}$; let $\mathcal{CH}(S)$ be the convex hull of S, and $\mathcal{T}(S)$ a triangulation of S, then the *extended triangulation* is given by

$$\mathcal{T}(S') = \mathcal{T}(S) \cup \{(f, \infty) \mid f \text{ facet of } \mathcal{CH}(S)\}. \tag{2.5}$$

In addition to $\mathcal{T}(S)$, every facet on the boundary of the convex hull $\mathcal{CH}(S)$ is connected to the point ∞; such connections produce new simplices incident to ∞, which are called *infinite simplices*. In contrast with $\mathcal{T}(S)$, $\mathcal{T}(S')$ has the convenient property that there are exactly $d + 1$ simplices adjacent to each simplex in $\mathcal{T}(S')$; $\mathcal{T}(S')$ can also be seen as a triangulation of the topological hypersphere.

2.2.3 Delaunay

A triangulation of a set of points is a *Delaunay triangulation* if no point in the set is inside the circumsphere of any cell in the triangulation [182, 37]; see Figure 2.5. Several Delaunay
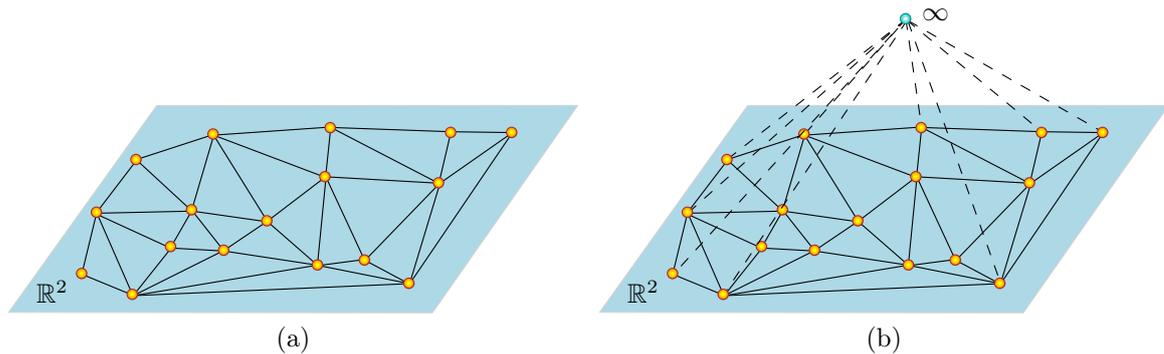
Figure 2.4: **Infinite vertex.** *Representation: (a) without infinite vertex; (b) with infinite vertex.*

triangulation construction algorithms have been described in the literature. Many of them are appropriate in the *static* setting [197, 119], where the points are fixed and known in advance. There is also a variety of so-called *dynamic* algorithms [127, 93, 100], in which the points are fixed and the triangulation is maintained under point insertions or deletions.

2.2.4 Certificates

Among several instances of predicates on computational geometry, in this thesis, we are specially interested in two predicates related to (Delaunay) triangulations: the *orientation predicate* and the *empty-sphere predicate*; this section presents both of them.

When one or more predicates are used to evaluate whether a geometric data structure is valid or invalid, we denote by *certificate* each of those predicates. By convention, we assume that a certificate is said to be *valid* when it is positive.

A triangulation lying in $\mathbb{R}^d$ can be checked to be valid by using the *orientation certificate* [95]: For each cell of the triangulation, every $d + 1$ vertices have the same orientation. (The definition of a valid triangulation is presented in Section 2.2.1.) Therefore, this certificate is applied to the sequence of $d + 1$ points $p_1 = (x_{1\,1}, \ldots, x_{1\,d}), \ldots, p_{d+1} = (x_{d+1\,1}, \ldots, x_{d+1\,d})$ belonging to a cell of the triangulation. The orientation certificate in $\mathbb{R}^d$ is the sign of the determinant of the following matrix:

$$\begin{pmatrix} x_{1\,1} & \ldots & x_{1\,d} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{d+1\,1} & \ldots & x_{d+1\,d} & 1 \end{pmatrix}. \tag{2.6}$$

See Figure 2.6(a) and 2.6(b) for an illustration of a valid and invalid set of cells in two dimensions. By convention, vertices of a cell are stored in an order such that the evaluation of the orientation predicate on these vertices gives a positive sign; see Figure 2.3. When all the cells of a triangulation lying in $\mathbb{R}^d$ are correctly oriented (i.e., positively oriented), we say that the triangulation is *embedded* in $\mathbb{R}^d$; or simply *embedded*, when there is no ambiguity.

A triangulation embedded in $\mathbb{R}^d$ can be checked to be Delaunay by using the *empty-sphere certificate* [95]: For each pair of adjacent cells of the triangulation (including infinite cells), the hypersphere passing through the $d + 1$ vertices of a cell on one side does not
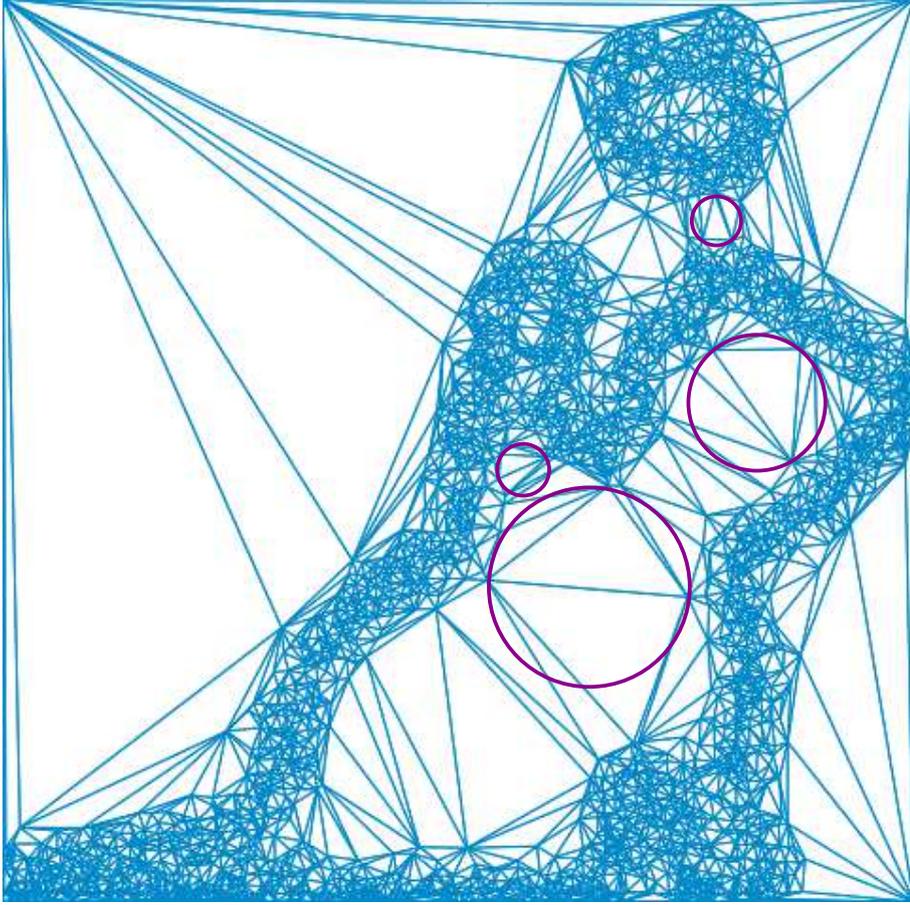
Figure 2.5: **Delaunay triangulation.**

contain the vertex on the other side. Therefore, this certificate is applied to the sequence of $d + 2$ points $p_1 = (x_{1\,1}, \ldots, x_{1\,d}), \ldots, p_{d+2} = (x_{d+2\,1}, \ldots, x_{d+2\,d})$ belonging to a pair of adjacent cells of the triangulation. The empty-sphere certificate in $\mathbb{R}^d$ is the sign of the determinant of the following matrix:

$$\begin{pmatrix} x_{1\,1} & \cdots & x_{1\,d} & \sum_{i=1}^{d} x_{1\,i}^2 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{d+2\,1} & \cdots & x_{d+2\,d} & \sum_{i=1}^{d} x_{d+2\,i}^2 & 1 \end{pmatrix}. \tag{2.7}$$

This is equivalent to the orientation certificate in $\mathbb{R}^{d+1}$, when the points are lifted on the unit paraboloid $\Pi = \{(x_1, \ldots, x_{d+1}) \in \mathbb{R}^{d+1} \mid x_{d+1} = \sum_{i=1}^{d} x_d^2\}$. See Figure 2.6(c) and 2.6(d) for an illustration of a valid and invalid pair of adjacent cells in two dimensions. When one or both cells of the pair of adjacent cells are infinite, then there is a total of $d + 1$ finite points, and the empty-sphere certificate boils down to an orientation certificate: The sphere passing through the boundary facet and the "point at infinity" degenerates into a plane, and the certificate becomes equivalent to verifying whether a point lies on a particular side of that plane. Two situations should be considered here: (i) if the number of infinite cells is one, then the empty-sphere certificate is automatically verified by the validity of the triangulation; (ii) otherwise, the empty-sphere certificate becomes a *reversed orientation certificate*, as shown in Figure 2.7.
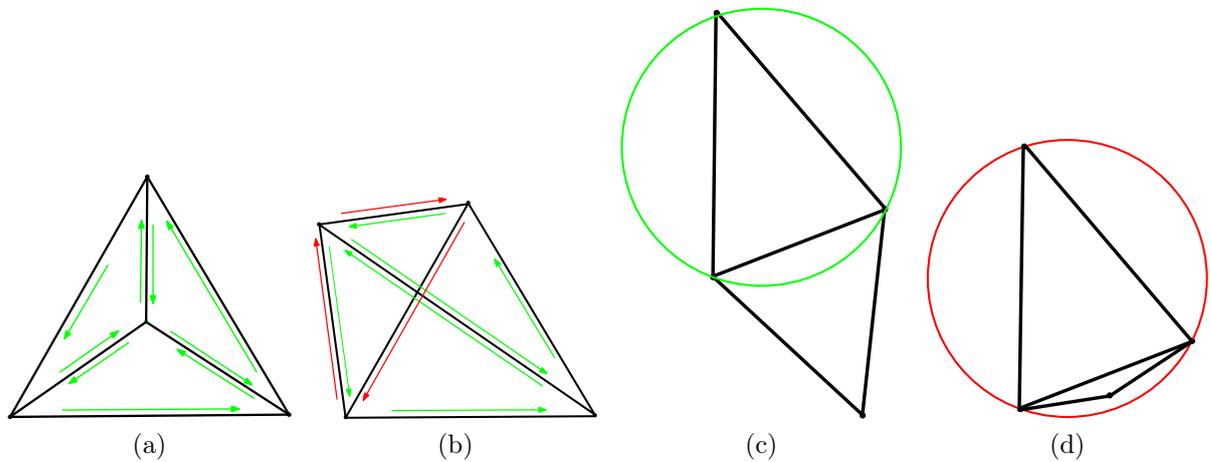
Figure 2.6: **Orientation and empty-sphere certificates.** *(a) all the cells are oriented in the same way (counter-clockwise), the orientation certificate is valid; (b) the cell with the red arrows is oriented incorrectly (clockwise), the orientation certificate fails; (c) circumscribed spheres (circles) are empty, the empty-sphere certificate is valid. (d) circumscribed spheres contain a point, the empty-sphere certificate is invalid. Semi-static and dynamic filtering in both two and three dimensions are implemented for these certificates in* CGAL *[137, 59].*
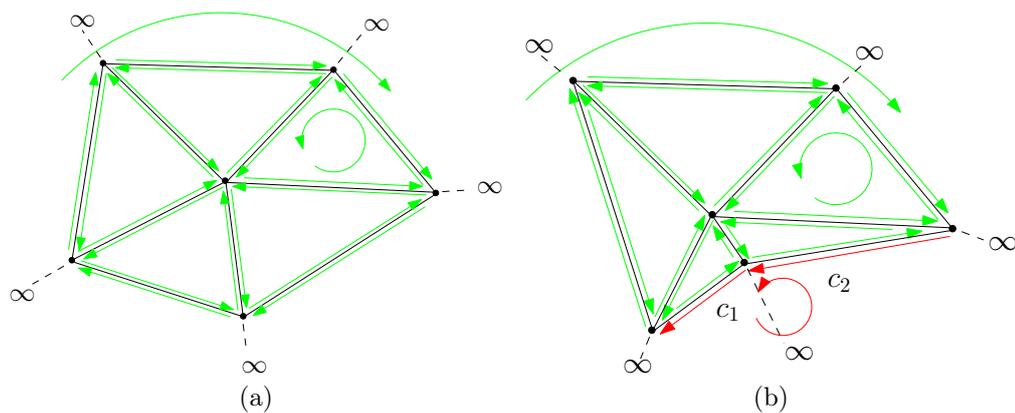
Figure 2.7: **Empty-sphere certificates on infinite cells.** *Orientation of finite cells are counter-clockwise: (a) orientation of three consecutive vertices of the convex hull is clockwise, the empty-sphere certificate is valid; and (b) there are three consecutive vertices of the convex hull with the same orientation as the finite cells, the empty-sphere certificate acting on the infinite cells c_1 and c_2 is invalid.*

Part I

Delaunay triangulations on the sphere

Chapter 3

Robust and Efficient Delaunay Triangulations of Points On or Close to a Sphere

---

"Le croissant de la lune, constamment dirigé vers le soleil, indique évidemment qu'elle en emprunte sa lumière." — Pierre-Simon Laplace.[1]

This chapter is a joint work with Manuel Caroli, Sebastien Loriot, Olivier Rouiller, Monique Teillaud and Camille Wormser.

- M. Caroli, P. M. M. de Castro, S. Loriot, O. Rouiller, M. Teillaud and C. Wormser. Robust and Efficient Delaunay Triangulations of Points on Or Close to a Sphere. In SEA '10:*Proceedings 9th International Symposium on Experimental Algorithms*, pages 462–473, 2010. (Also available as: Research Report 7004, INRIA, 2009.)

The CGAL project [5] provides users with a public discussion mailing list, where they are invited to post questions and express their needs. There are recurring requests for a package computing the Delaunay triangulation of points on a sphere or its dual, the Voronoi diagram. This is useful in many domains such as geophysics [118] (see Figure 3.1(a)), geographic information systems (see Figure 3.9), information visualization [154] (see Figure 3.1(b)), or structural molecular biology, to name a few.

An easy and standard solution to the problem of computing such a Delaunay triangulation consists in constructing the 3D convex hull of the points: They are equivalent [58, 215]. The convex hull is one of the most popular structures in computational geometry [82, 51]; optimal algorithms and efficient implementations are available [66, 3, 4].

Another fruitful way to compute Delaunay on a sphere consists of reworking known algorithms designed for computing triangulations in $\mathbb{R}^2$. Renka adapts the distance in the

---

[1]The crescent of the moon being always directed towards the sun, indicates obviously that she borrows her light from that luminary.

<div align="center">(a)   (b)</div>
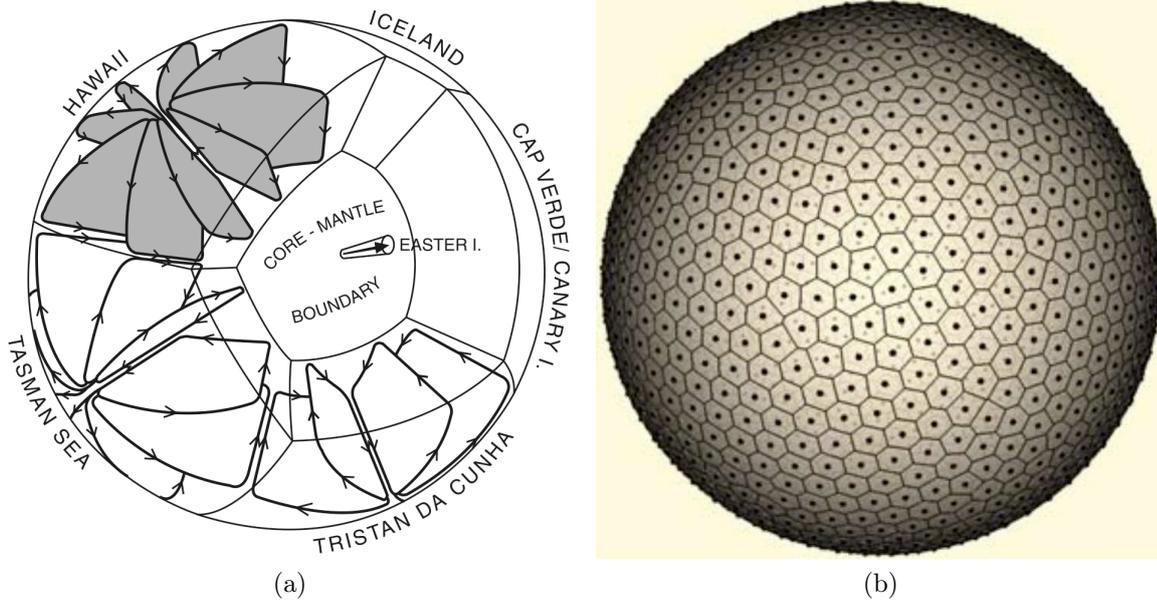
Figure 3.1: **Definitions.** *(a) Geophysical Simulation (picture taken from Fohlmeister [118])., (b) Information Visualization (picture taken from Larrea et al. [154]).*

plane to a geodesic distance on a sphere and triangulates points on a sphere [184] through the well-known flipping algorithm for Delaunay triangulations in $\mathbb{R}^2$ [155]. As a by-product of their algorithm for arrangements of circular arcs, Fogel et al. can compute Voronoi diagrams of points lying exactly on the sphere [116]. Using two inversions allows Na et al. to reduce the computation of a Voronoi diagram of sites on a sphere to computing two Voronoi diagrams in $\mathbb{R}^2$ [172]; however, to the best of our knowledge, no implementation is available. Note that this method assumes that data points are lying exactly on a sphere.

As we are motivated by applications, we take practical issues into account carefully. While data points lying exactly on the sphere can be provided either by using Cartesian coordinates represented by a number type capable of handling algebraic numbers exactly, or by using spherical coordinates, in practice data-sets in Cartesian coordinates with double precision are most common. In this setting, the data consists of rounded points that do not exactly lie on the sphere, but close to it.

Our Contributions. In Section 3.3, we propose two different ways to handle such rounded data. Both approaches adapt the well-known incremental algorithm [55] to the case of points on, or close to the sphere. It is important to notice that, even though data points are rounded, we follow the exact geometric computation paradigm pioneered by Yap [221]. Indeed, it is now well understood that simply relying on floating-point arithmetic for algorithms of this type is bound to fail (see [147] for instance).

The first approach (Section 3.3.1) considers as input the projections of the rounded-data points onto the sphere. Their coordinates are algebraic numbers of degree two. The approach computes the Delaunay triangulation of these points exactly lying on the sphere.

The second approach (Section 3.3.2) considers circles on the sphere as input. The radius of a circle (which can alternatively be seen as a *weighted* point) depends on the distance of the corresponding point to the sphere. The approach computes the weighted Delaunay triangulation of these circles on the sphere, also known as the *regular* triangu-

lation, which is the dual of the Laguerre Voronoi diagram on the sphere [215] and the convex hull of the rounded-data points.

These interpretations of rounded data presented in this work are supported by the space of circles [44, 96] (Section 3.2).

We implemented both approaches, taking advantage of the genericity of CGAL. In Section 3.4, we present experimental results on very large data-sets, showing the efficiency of our approaches. We compare our code to software designed for computing Delaunay triangulations on the sphere, and to convex hull software [138, 179, 3, 2, 4, 184, 115]. The performance, robustness, and scalability of our approaches express their added value.

3.1 Definitions and Notation

Let us first recall the definition of the *regular triangulation* in $\mathbb{R}^2$, also known as *weighted Delaunay triangulation*. A circle c with center $p \in \mathbb{R}^2$ and squared radius r^2 is considered equivalently as a *weighted point* and is denoted by $c = (p, r^2)$. The *power product* of $c = (p, r^2)$ and $c' = (p', r'^2)$ is defined as $pow(c, c') = \|pp'\|^2 - r^2 - r'^2$, where $\|pp'\|$ denotes the Euclidean distance between p and p'. Circles c and c' are orthogonal if and only if $pow(c, c') = 0$. If $pow(c, c') > 0$ (i.e., the disks defined by c and c' do not intersect, or the circles intersect with an angle strictly smaller than $\frac{\pi}{2}$), we say that c and c' are *suborthogonal*. If $pow(c, c') < 0$, then we say that c and c' are *superorthogonal*; see Figure 3.2. Three circles whose centers are not collinear have a unique common orthogonal circle.
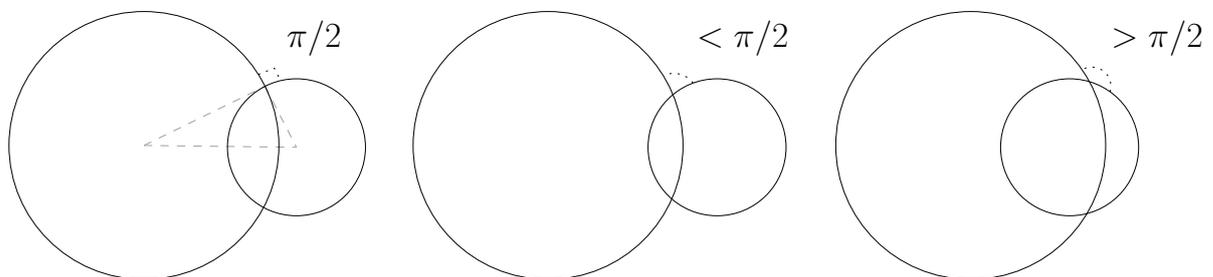
Figure 3.2: **Power product.** *From left to right: orthogonal ($pow(c, c') = 0$), suborthogonal ($pow(c, c') > 0$), and superorthogonal ($pow(c, c') < 0$) circles in $\mathbb{R}^2$.*

Let $\mathcal{S}$ be a set of circles. Given three circles of $\mathcal{S}$, $c_i = (p_i, r_i^2)$, $i = 1 \ldots 3$, whose centers are not collinear, let T be the triangle whose vertices are the three centers p_1, p_2, and p_3. We define the *orthogonal circle* of T as the circle that is orthogonal to the three circles c_1, c_2, and c_3. T is said to be *regular* if for any circle $c \in \mathcal{S}$, the orthogonal circle of T and c are not superorthogonal. A *regular triangulation* $\mathcal{RT}(\mathcal{S})$ is a partition of the convex hull of the centers of the circles of $\mathcal{S}$ into regular triangles formed by these centers; see Figure 3.3 for an example. The dual of the regular triangulation is known as the *power diagram*, *weighted Voronoi diagram*, or *Laguerre diagram*.

If all radii are equal, then the power test reduces to testing whether a point lies inside, outside, or on the circle passing through three points; the regular triangulation of the circles is the Delaunay triangulation $\mathcal{DT}$ of their centers.
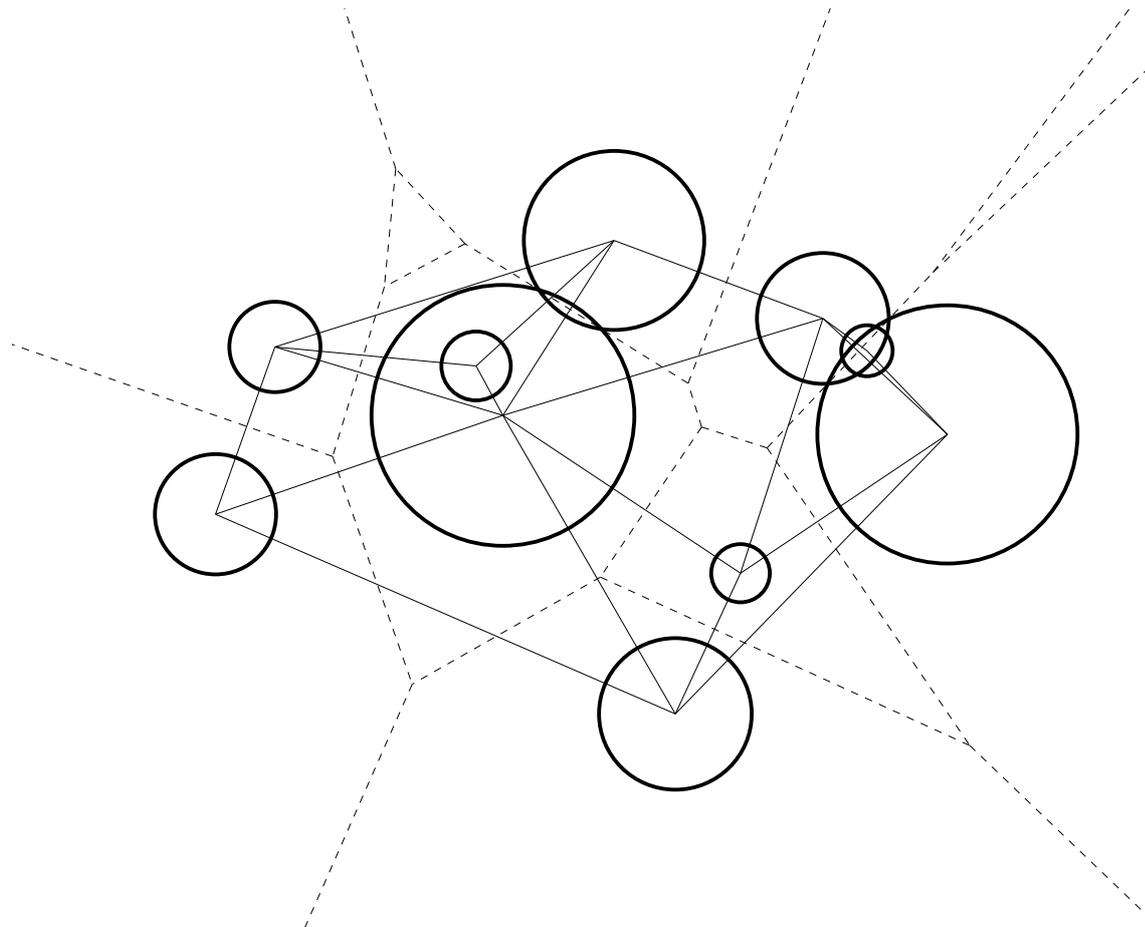
Figure 3.3: **Regular triangulation.** *Regular triangulation of a set of circles in the plane (their power diagram is shown dashed)*

More background can be found in [36]. We refer the reader to standard textbooks for algorithms computing Delaunay and regular triangulations [82, 51].

This definition generalizes in a natural manner to the case of circles lying on a sphere $\mathbb{S}$ in $\mathbb{R}^3$: Angles between circles are measured on the sphere, triangles are drawn on the sphere, their edges being arcs of great circles. As can be seen in the next section, the space of circles provides a geometric presentation showing without any computation that the regular triangulation on $\mathbb{S}$ is a convex hull in $\mathbb{R}^3$ [215].

In the sequel, we assume that $\mathbb{S}$ is given by its center, having rational coordinates (we take the origin O without loss of generality), and a rational squared radius R^2. This is also how spheres are represented in CGAL.[2]

3.2   Space of Circles

Computational geometers are familiar with the classic idea of lifting up sites from the Euclidean plane onto the unit paraboloid Π in $\mathbb{R}^3$ [37]. We quickly recall the notion

---

[2]We mention rational numbers to simplify the presentation. CGAL allows more general number types that provide field operations: $+, -, \times, /$.

Algorithm 23

of *space of circles* here and refer to the literature for a more detailed presentation [96]. In this lifting, points of $\mathbb{R}^3$ are viewed as circles of $\mathbb{R}^2$ in the space of circles: A circle $c = (p, r^2)$ in $\mathbb{R}^2$ is mapped by π to the point $\pi(c) = (x_p, y_p, x_p^2 + y_p^2 - r^2) \in \mathbb{R}^3$. A point of $\mathbb{R}^3$ lying respectively outside, inside, or on the paraboloid Π represents a circle with respectively positive, imaginary, or null radius. The circle c in $\mathbb{R}^2$ corresponding to a point $\pi(c)$ of $\mathbb{R}^3$ outside Π is obtained as the projection onto $\mathbb{R}^2$ of the intersection between Π and the cone formed by lines through $\pi(c)$ that are tangent to Π; this intersection is also the intersection of the polar plane $P(c)$ of $\pi(c)$ with respect to the quadric Π.

Points lying respectively on, above, below $P(c)$ correspond to circles in $\mathbb{R}^2$ that are respectively orthogonal, suborthogonal, superorthogonal to c. The predicate $pow(c, c')$ introduced above is thus equivalent to the orientation predicate in $\mathbb{R}^3$ that tests whether the point $\pi(c')$ lies on, above or below the plane $P(c)$. If c is the common orthogonal circle to three input circles c_1, c_2, and c_3 (where $c_i = (p_i, r_i^2)$ for each i), then $pow(c, c')$ is the orientation predicate of the four points $\pi(c_1), \pi(c_2), \pi(c_3), \pi(c')$ of $\mathbb{R}^3$. It can be expressed as

$$\text{sign} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_{p_1} & x_{p_2} & x_{p_3} & x_{p'} \\ y_{p_1} & y_{p_2} & y_{p_3} & y_{p'} \\ z_{p_1} & z_{p_2} & z_{p_3} & z_{p'} \end{vmatrix}, \tag{3.1}$$

where $z_{p_i} = x_{p_i}^2 + y_{p_i}^2 - r_i^2$ for each i and $z_{p'}^2 = x_{p'}^2 + y_{p'}^2 - r'^2$. It allows to relate Delaunay or regular triangulations in $\mathbb{R}^2$ and convex hulls in $\mathbb{R}^3$ [37], while Voronoi diagrams in $\mathbb{R}^2$ are related to upper envelopes of planes in $\mathbb{R}^3$.

Up to a projective transformation, a sphere in $\mathbb{R}^3$ can be used for the lifting instead of the usual paraboloid [44]. In this representation the sphere has a pole[3] and can be identified to the Euclidean plane $\mathbb{R}^2$. What we are interested in this chapter is the space of circles drawn on the sphere $\mathbb{S}$ itself, without any pole. This space of circles has a nice relation to the de Sitter space in Minkowskian geometry [80].

We can still construct the circle c on $\mathbb{S}$ that is associated to a point $p = \pi_{\mathbb{S}}(c)$ of $\mathbb{R}^3$ as the intersection between $\mathbb{S}$ and the polar plane $P_{\mathbb{S}}(p)$ of p with respect to the quadric $\mathbb{S}$; see Figure 3.4. Its center is the projection of p onto $\mathbb{S}$ and as above, imaginary radii are possible.[4] So, in the determinant in Eq.(3.1), x_{p_i}, y_{p_i}, and z_{p_i} (respectively $x_{p'}, y_{p'}, z_{p'}$) are precisely the coordinates of the points $p_i = \pi_{\mathbb{S}}(c_i)$ (respectively $p' = \pi_{\mathbb{S}}(p)$). This is extensively used in Section 3.3. Again, we remark that Delaunay and regular triangulations on $\mathbb{S}$ relate to convex hulls in 3D.

Interestingly, rather than using a convex hull algorithm to obtain the Delaunay or regular triangulation on the surface as usually done for $\mathbb{R}^2$ [37], we do the converse in the next section.

3.3 Algorithm

The incremental algorithm for computing a regular triangulation of circles on the sphere $\mathbb{S}$ is a direct adaptation of the algorithm in $\mathbb{R}^2$ [55]. Assume that $\mathcal{RT}_{i-1} = \mathcal{RT}(\{c_j \in \mathcal{S}, j = 1, \ldots, i-1\})$ has been computed.[5] The insertion of $c_i = (p_i, r_i^2)$ works as follows:

---

[3]See the nice treatment of infinity in [44].

[4]Remember that $\mathbb{S}$ is centered at O and has squared radius R^2.

[5]For the sake of simplicity, we assume that the center O of $\mathbb{S}$ lies in the convex hull of the data-set. So, we just initialize the triangulation with four dummy points that contain O in their convex hull.
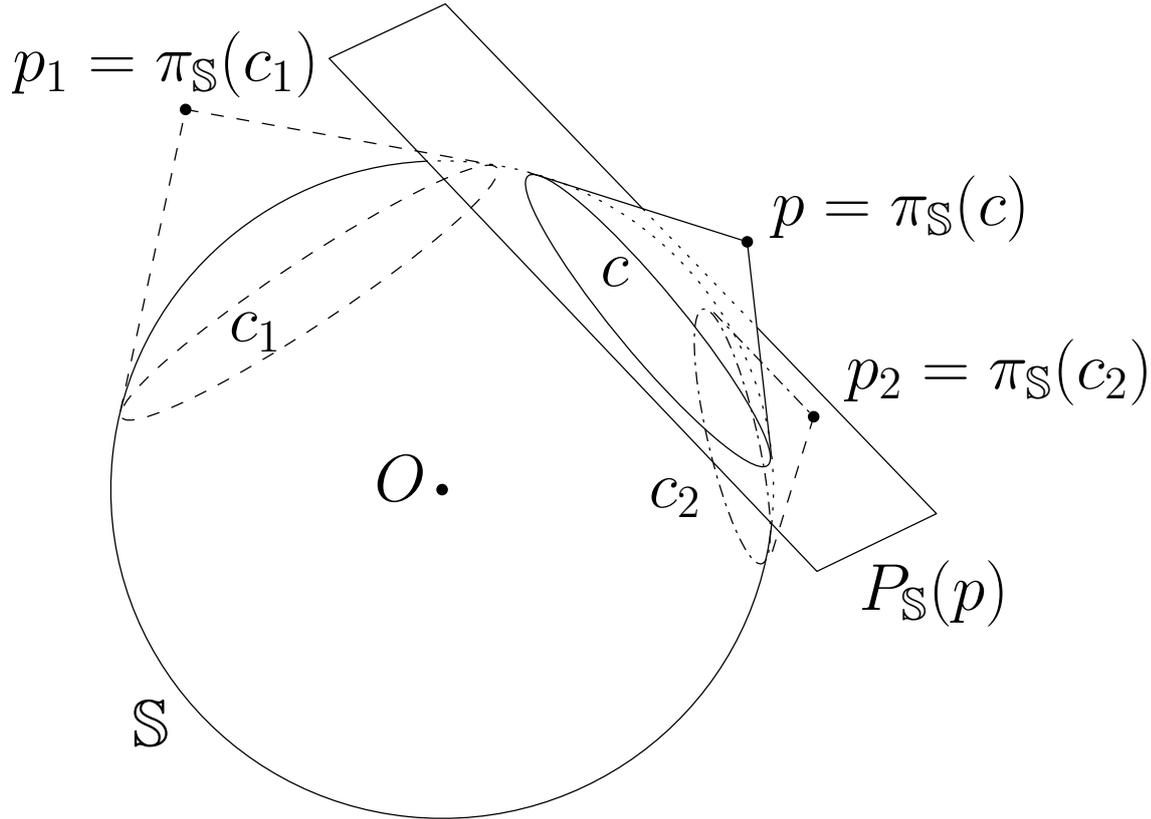
Figure 3.4: **Power product on the sphere.** c_1 *is suborthogonal to* c, c_2 *is superorthogonal to* c.

- locate p_i (i.e., find the triangle t containing p_i),
- if t is hiding p_i (i.e., if c_i and the orthogonal circle of t are suborthogonal) then stop; p_i is not a vertex of $\mathcal{RT}_i$. Note that this case never occurs for Delaunay triangulations.
- else (*i*) find all triangles whose orthogonal circles are superorthogonal to c_i and remove them; this forms a polygonal region that is star-shaped with respect to p_i;[6] (*ii*) triangulate the polygonal region just created by constructing the triangles formed by the boundary edges of the region and the point p_i.

Two main predicates are used by this algorithm:

The *orientation* predicate allows to check the orientation of three points p, q, and r on the sphere. (This predicate is used in particular to locate new points.) It is equivalent to computing the side of the plane defined by O, p, and q on which r is lying, i.e., the orientation of O, p, q, and r in $\mathbb{R}^3$.

The *power test* introduced in Section 3.1 boils down to an orientation predicate in $\mathbb{R}^3$, as seen in Section 3.2. (This predicate is used to identify the triangles whose orthogonal circles are superorthogonal to each new circle.)

The two approaches briefly presented in the introduction fall into the general framework of computing the regular triangulation of circles on the sphere. The next two sections precisely show how these predicates are evaluated in each approach.

---

[6]As previously noted for the edges of triangles, all usual terms referring to segments are transposed to arcs of great circles on the sphere.

Algorithm   25

3.3.1 First approach: using points on the sphere

In this approach, input points for the computation are chosen to be the projections on $\mathbb{S}$ of the rounded points of the data-set with rational coordinates. The three coordinates of an input point are thus algebraic numbers of degree two lying in the same extension field of the rationals.

In this approach weights, or equivalently radii if circles, are null. The power test consists in this case in answering whether a point s lies inside, outside,[7] or on the circle passing through p, q, and r on the sphere. Following Section 3.2, this is given by the orientation of p, q, r, and s, since points on the sphere are mapped to themselves by $\pi_\mathbb{S}$.

The difficulty comes from the fact that input points have algebraic coordinates. The coordinates of two different input points on the sphere are in general lying in different extensions. Then the 3D orientation predicate of p, q, r, and s given by Eq.(3.1) is the sign of an expression lying in an algebraic extension of degree 16 over the rationals, of the form $a_1\sqrt{\alpha_1} + a_2\sqrt{\alpha_2} + a_3\sqrt{\alpha_3} + a_4\sqrt{\alpha_4}$ where all a's and α's are rational. Evaluating this sign in an exact way allows to follow the exact computation framework ensuring the robustness of the algorithm.

Though software packages offer exact operations on general algebraic numbers [6, 8], they are much slower than computing with rational numbers. The sign of the above simple expression can be computed as follows:

–1– evaluate the signs of $A_1 = a_1\sqrt{\alpha_1} + a_2\sqrt{\alpha_2}$ and $A_2 = a_3\sqrt{\alpha_3} + a_4\sqrt{\alpha_4}$, by comparing $a_i\sqrt{\alpha_i}$ with $a_{i+1}\sqrt{\alpha_{i+1}}$ for $i = 1, 3$, which reduces after squaring to comparing two rational numbers,

–2– the result follows if A_1 and A_2 have the same signs,

–3– otherwise, compare A_1^2 with A_2^2, which is an easier instance of –1–.

To summarize, the predicate is given by the sign of polynomial expressions in the rational coordinates of the rounded-data points, which can be computed exactly using rational numbers only.

3.3.2 Second approach: using weighted points

In this approach, the regular triangulation of the weighted points is computed as described above. As in the previous approach, both predicates (orientation on the sphere and power test) reduce to orientation predicates on the data points in $\mathbb{R}^3$. Note that Section 3.2 shows that the weight of a point p is implicit, as it does not need to be explicitly computed throughout the entire algorithm.

Depending on the weights, some points can be hidden in a regular triangulation. We prove now that under some sampling conditions on the rounded data, there is actually no hidden point.

Lemma 9. *Let us assume that all data points lie within a distance δ from $\mathbb{S}$. If the distance between any two points is larger than $2\sqrt{R\delta}$, then no point is hidden.*

Proof. A point is hidden if and only if it is contained inside the 3D convex hull of the set of data points $\mathcal{S}$. Let p be a data point, at distance ρ from O. We have $\rho \in [R -$

---

[7]On $\mathbb{S}$, the interior (respectively exterior) of a circle c that is not a great circle of $\mathbb{S}$ corresponds to the interior (respectively exterior) of the half-cone in 3D, whose apex is the center of $\mathbb{S}$ and that intersects $\mathbb{S}$ along c.

$\delta, R + \delta]$. Denote by d_p the minimum distance between p and the other points. If $d_p > \sqrt{(R + \delta)^2 - \rho^2}$, the set $B(O, R + \delta) \setminus B(p, d_p)$ is included in the half-space $H^+ = \{q : \langle q - p, O - p \rangle > 0\}$. Under these conditions, all other points belong to H^+ and p is not inside the convex hull of the other points. It follows that if the distance between any two data points is larger than $\sup_\rho \sqrt{(R + \delta)^2 - \rho^2} = 2\sqrt{R\delta}$, no point is hidden. $\square$

Let us now assume we use double precision floating-point numbers as defined in the IEEE standard 754 [9, 124]. The mantissa is encoded using 52 bits. Let γ denote the worst error, for each Cartesian coordinate, done while rounding a point on $\mathbb{S}$ to the nearest point whose coordinates can be represented by double precision floating-point numbers. Let us use the standard term $\mathrm{ulp}(x)$ denoting the gap between the two floating-point numbers closest to the real value x [169] (also discussed in Section 2.1.3). Assuming again that the center of $\mathbb{S}$ is O, one has $\gamma \leq \mathrm{ulp}(R) \leq 2^{-52+\lfloor \log_2(R) \rfloor} \leq 2^{-52}R$. Then, δ in the previous lemma is such that $\delta \leq \sqrt{3/4}\gamma < 2^{-52}R$. Using the result of the lemma, no point is hidden in the regular triangulation as soon as the distance between any two points is greater than $2^{-25}R$.

Note that this approach can be used as well to compute the convex hull of points that are not close to a sphere: The center of the sphere can be chosen at any point inside a tetrahedron formed by any four non-coplanar data points.

3.4   Implementation and Experiments

Both approaches presented in Section 3.3 have been implemented in C++, based on the CGAL package that computes triangulations in $\mathbb{R}^2$. The package introduces an infinite vertex in the triangulation to compactify $\mathbb{R}^2$ (discussed in Section 2.2.2). Thus the underlying combinatorial triangulation is a triangulation of the topological sphere. This allows us to reuse the whole combinatorial part of CGAL 2D triangulations [180] without any modification. However, the geometric embedding of CGAL 2D triangulation [223], bounded to $\mathbb{R}^2$, must be modified by removing any reference to the infinite vertex. A similar work was done to compute triangulations in the 3D flat torus [61, 62], reusing the CGAL 3D triangulation package [179] as much as possible.

The genericity offered in CGAL allows the encapsulation of geometric predicates; such genericity is obtained through the mechanism of *traits classes* [171]. Traits classes allow us to easily use exactly the same algorithm for both approaches. We implement two different traits classes, each one corresponding to a distinct approach; see Figure 3.5.

To display the triangulation and its dual, the code is interfaced with the CGAL 3D spherical kernel [84, 83], which provides primitives on circular arcs in 3D. The vertices of the triangulations shown are the projections on the sphere of the rounded-data points. The circular arcs are drawn on the surface of the sphere; see Figures 3.7, 3.8, and 3.9.

We compare the running time of our approaches with several available software packages on a MacBook Pro 3,1 equipped with a 2.6 GHz Intel Core 2 processor and 2GB 667 MHz DDR2 SDRAM[8]; see Figure 3.6. We consider large sets of random data points[9] (up to 2^{23} points) on the sphere, rounded to double coordinates. Figure 3.7 indicates running

---

[8] Further details: MAC OS X version 10.5.7, 64 bits; compiler `g++` `4.3.2` with -O3 and -DNDEBUG flags, `g77` `3.4.3` with -O3 for Fortran. All running times mentioned exclude time used by input/output.

[9] generated by `CGAL::Random_points_on_sphere_3`
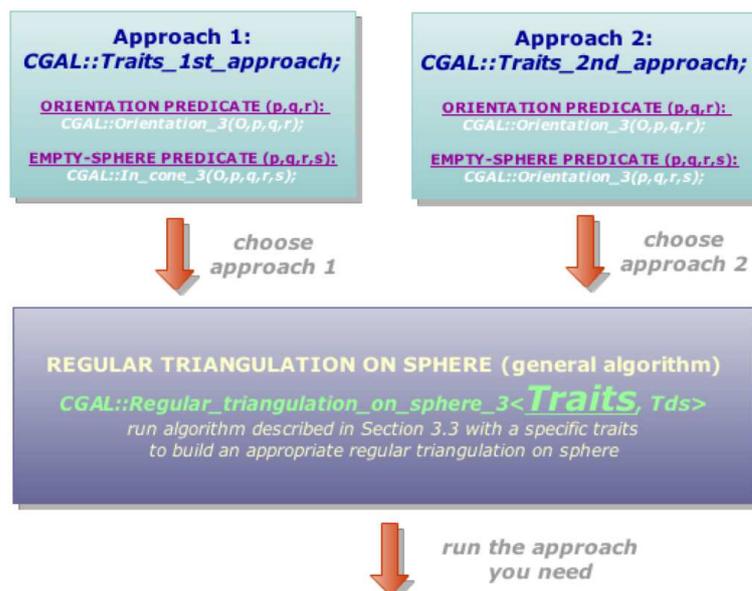
Figure 3.5: **Run both approaches with one single algorithm.**

times on some real-life data. Each experiment was repeated 30 times, and the average is taken.

Graph 1st of Figure 3.6 shows the results of our first approach. We coded a traits class implementing the exact predicates presented in Section 3.3.1, together with semi-static and dynamic filtering [158]. The non-linear behavior of the running time is due to the fact that our semi-static filters hardly ever fail for less than 2^{13} points, and almost always fail for more than 2^{18} points.

Graph 2nd shows the results of the second approach. One of the predefined kernels[10] of CGAL provides us directly with an exact implementation of the predicates, filtered both semi-statically and dynamically. In our experiments we have observed that no point is hidden with such distributions, even when the data-set is large, which confirms in practice the discussion of Section 3.3.2.

The CGAL 3D Delaunay triangulation (graph DT3) [179], with the same CGAL kernel, also provides this convex hull as a by-product. We insert the center of the sphere to avoid penalizing this code with too many predicate calls on five cospherical points that would always cause filters to fail.

For these three approaches, 3D spatial sorting reduces the running time of the location step of point insertion [88, 60].

If the data points are lying exactly on a sphere, their Delaunay Triangulation can be extracted from an arrangement of geodesic arcs as computed by the code of Fogel and Setter [115, 116]. Since it is not the main purpose of their algorithm, the running times are not comparable: close to 600 seconds for 2^{12} points. Note however that the code is preliminary and has not been fully optimized yet. No graph is shown.

We consider the following two software packages computing a convex hull in 3D,[11] for

---

[10]precisely `CGAL::Exact_predicates_inexact_constructions_kernel`

[11]The plot does not show the results of the CGAL 3D convex hull package [138] because it is much slower than all other methods (roughly 500 times slower than QHULL).
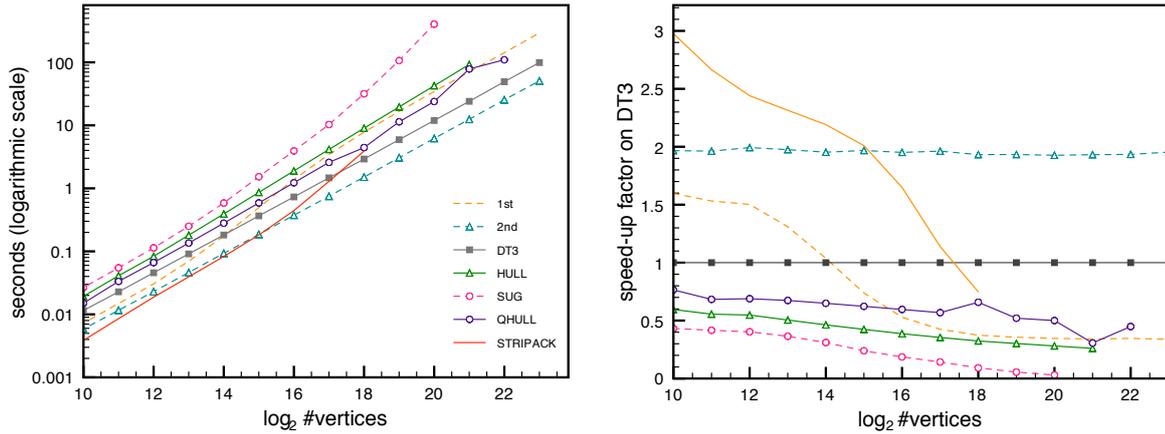
Figure 3.6: **Comparative benchmarks.** *The programs were aborted when their running time was above 10 minutes (HULL, SUG, QHULL) or in case of failure (STRIPACK).*

which the data points are first rounded to points with integer coordinates. Predicates are evaluated exactly using single precision computations.

Graph HULL corresponds to the code [3] of Clarkson, who uses a randomized incremental construction [75] with an exact arithmetic on integers [74].

Graph SUG shows the running times of Sugihara's code in Fortran [2, 215].

Graph QHULL shows the performance of the famous Qhull package of Barber et al. [4] when computing the 3D convex hull of the points. The option we use handles round-off errors from floating point arithmetic by merging facets of the convex hull when necessary. The convex hull of points situated close to the sphere contains in practice all the input points; see Lemma 9. In this situation QHULL is clearly outperformed by the second approach. However, QHULL can be about twice faster than our second approach when almost all the input points are hidden.

Renka computes the triangulation with an algorithm similar to our first approach, but his software STRIPACK, in Fortran, uses approximate computations in double [184]. Consequently, it performs quite well on random points (better than our implementations for small random data-sets), but it fails on some data-sets: Using STRIPACK, we did not manage to compute a triangulation of more than 2^{19} random points (it returns an error flag). The same occurred for the inputs used to produce Figures 3.7, 3.8, and 3.9. Our implementations handle arbitrary data sets.

To test for exactness we devised a point set that is especially hard to triangulate because it yields many very flat triangles in the triangulation. This point set is defined as

$$\mathcal{S}_n = \left\{ \begin{pmatrix} \cos\theta\sin\phi \\ \sin\theta\sin\phi \\ \cos\phi \end{pmatrix} \middle| {\scriptstyle \theta \in \left\{ 0, \frac{\pi}{n}, \ldots, \frac{(n-1)\pi}{n}, \pi \right\}, \phi = \frac{(\theta^2+1)}{\pi^2} } \right\} \cup \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, -\frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\},$$

In Figure 3.8 we show the Delaunay triangulation of $\mathcal{S}_{250}$ and the Voronoi diagram of $\mathcal{S}_{100}$.

In Table 3.1, we compare the memory usage of our two approaches, the 3D Delaunay triangulation, and Qhull. The given figures given in bytes per processed vertex (bppv) and averaged over several data-sets of size larger than 2^{16}.
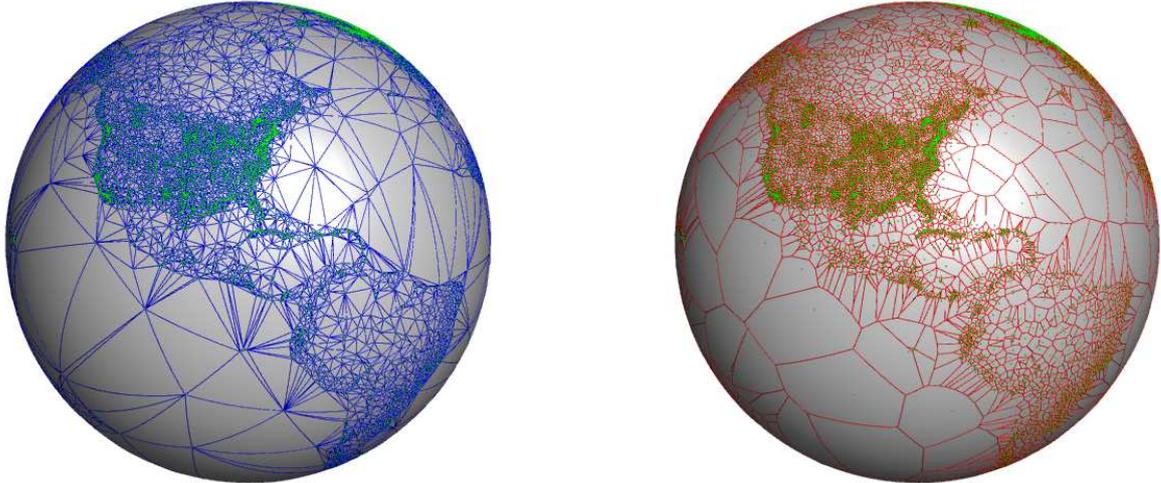
Figure 3.7: **Weather stations.** *Delaunay triangulation (left) and Voronoi diagram (right) of 20,950 weather stations all around the world. Data and more information can be found at* `http://www.locationidentifiers.org/`. *Our second approach computes the result in 0.14 seconds, while Qhull needs 0.35 seconds, and the first approach 0.57 seconds.* STRIPACK *fails on this data-set.*

Approach	1st	2nd	DT3	QHULL
Bytes per Vertex	**113**	**113**	174	288

Table 3.1: **Memory usage.** *These measurements are done with* `CGAL::Memory_sizer` *for the first approach, second approach and for the 3D Delaunay triangulation. For the Qhull package the measurement is done with the -Ts option, taking into account the memory allocated for facets and their normals, neighbor and vertex sets.*

3.5   Conclusion

The results show that our second approach yields better timings and memory usage than all the other tested software packages for large data-sets, while being fully robust. This justifies a typical phenomenon: The well-designed specialized solution outperforms the more general one. Here the specialized one is our second approach, and the general one is the Delaunay triangulation 3D computation from which the 3D convex hull is extracted.

The first approach is slower but still one of the most scalable. It exactly computes the triangulation for input points with algebraic coordinates lying on the sphere, and thus ensures that in any case all points appear in the triangulation. It is the only one to do so within reasonable time and thus being useful for real-world applications.

One important news this chapter brings — and yet, a very intuitive one — is that: (i) when most of the input points lie on their convex hull, then a two-dimensional structure is more adequate than a three-dimensional one (for the incremental construction); (ii) in the converse, then a three-dimensional is more adequate. Although we have not experimented in higher-dimensions, we think that this same phenomenon repeats; this leads to the question below.
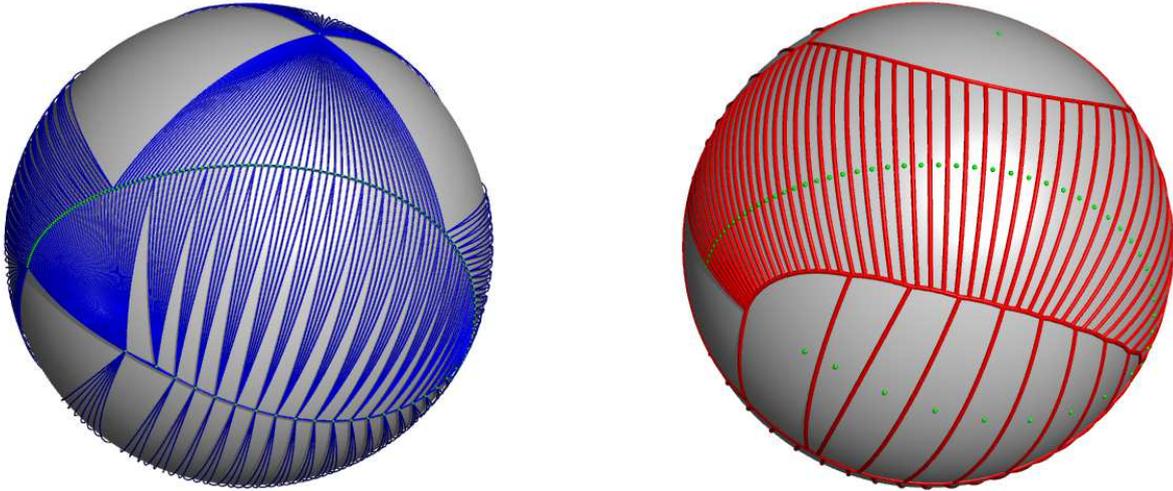
Figure 3.8: **Hard cases with skinny triangles.** *Delaunay triangulation of $\mathcal{S}_{250}$ (left), Voronoi diagram of $\mathcal{S}_{100}$ (right).* STRIPACK *fails for e.g., $n = 1,500$.*

3.6   Open Problem

Problem 10. *Does modifying the d-dimensional Delaunay triangulation of Boissonnat et al. [49] in order to handle Regular triangulations, improve the $(d+1)$-dimensional convex hull construction when most of the input points are on their convex hull?*

We believe that an extension of the first approach to higher dimensions would probably fail, because the orientation predicate computation cannot be simplified as direct computations on rationals. However, the second approach does not seem to be highly impacted by the dimension (as long as the dimension remains "reasonably" small); leaving some hope for future works.
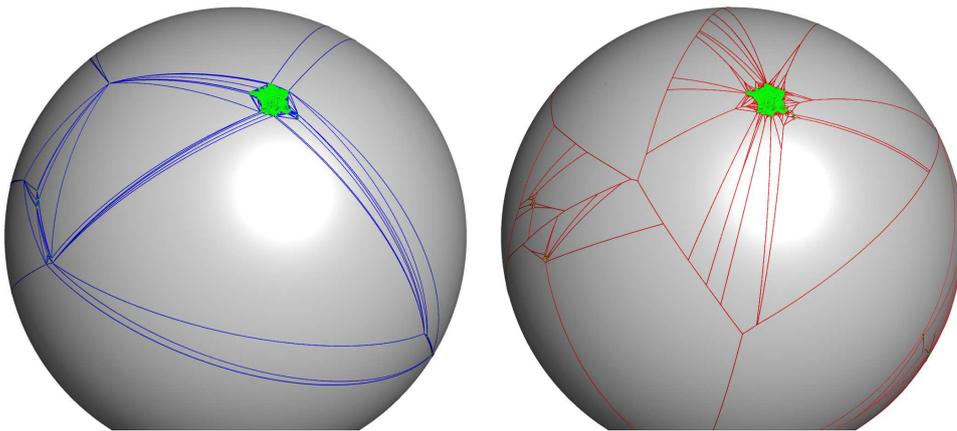
Figure 3.9: **Post-offices in France.** Delaunay triangulation (left) and Voronoi diagram (right) of the 9,031 post-offices in France (including Overseas Departments and Territories).

Part II

Relocations on a Delaunay triangulation

Chapter 4

State of the Art: Moving Points on Triangulations

"As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality." — *Albert Einstein.*

- P. M. M. de Castro and O. Devillers. State of the Art: Updating Delaunay Triangulations for Moving Points. Research Report 6665, INRIA, 2008.

Maintaining a triangulation of a set of points is the basis of many applications. In several of these applications, the point set evolves over time. Thus, the vertices of the triangulation — defined by input data points — are moving. This happens for instance in data clustering [136, 30], mesh generation [87], re-meshing [218, 23], mesh smoothing [25], mesh optimization [24, 216, 67], to name a few.

There are two distinct contexts, when looking at point relocations in a triangulation: (i) if points move continuously and we want to keep track of all topological changes, we call such a context a *kinetic relocation context* [129, 187]; (ii) if we are only interested in the triangulation at some discrete time stamps, we call such a context a *time stamp relocation context* [128, 187].

This chapter briefly describes the state of the art of point relocations in a triangulation, both in a kinetic relocation context (Section 4.1) and in a time stamp relocation context (Section 4.2). Section 4.1 focuses on theoretical aspects of point relocations, whereas Section 4.2 focuses on practical aspects of point relocations. Delaunay triangulations are emphasized, and several fundamentals described in this chapter are useful in Chapter 5.

4.1 Kinetic Relocation Context

In theory, maintaining a data structure of moving points is classically formalized through two very important concepts: (i) *kinetic data structures* [129], and (ii) *Davenport-Schinzel*

sequences [18]. The first concept substantiates a data structure where primitives move; and the second concept can be used to bound the complexity of such data structures.

4.1.1   Davenport-Schinzel Sequences

An (n, s) Davenport-Schinzel sequence, for positive integers n and s, is a sequence of n distinct symbols with the properties that no two adjacent elements are equal, and that it does not contain, as a subsequence, any alternations $a \cdots b \cdots a \cdots b$ of length $s + 2$ between two distinct symbols a and b. Here are some examples of such sequences for $n = 5$ and $s = 1, 2, 3$:

$$
\begin{aligned}
s = 1 &\rightarrow 1, 2, 3, 4, 5 \\
s = 2 &\rightarrow 1, 2, 3, 4, 5, 4, 3, 2, 1 \\
s = 3 &\rightarrow 1, 2, 1, 3, 1, 3, 2, 4, 5, 4, 5, 2, 3
\end{aligned}
$$

The size of the largest (n, s) Davenport-Schinzel sequence is denoted by $\lambda_s(n)$; besides, the ratio between the size of the largest (n, s) Davenport-Schinzel sequence and the number of symbols is given by $\beta_s(n) = \lambda_s(n)/n$. Bounds for $\lambda_s(n)$, for $s = 1$ and $s = 2$ are trivial ($\lambda_1(n) \leq n$, and $\lambda_2(n) \leq 2n - 1$). For $s = 3$, $\lambda_3(n) = O(n\alpha(n))$ [135, 150], where α is the inverse Ackermann function.[1] And for any arbitrary positive integer s, the best bounds up to 2009 are due to Nivasch [174]:[2]

$$
\lambda_s(n) \leq \begin{cases} n \cdot 2^{(1/t!)\alpha(n)^t + O\left(\alpha(n)^{t-1}\right)}, & s \geq 4 \text{ even;} \\ n \cdot 2^{(1/t!)\alpha(n)^t \log_2 \alpha(n) + O\left(\alpha(n)^t\right)}, & s \geq 3 \text{ odd;} \end{cases} \tag{4.1}
$$

where $t = \lfloor (s - 2)/2 \rfloor$. For any constant s, an easier bound (and less tight) is $\lambda_s(n) = O(n \log n)$ that is $\beta_s(n) = O(\log n)$.

Theorem 11 proved by Hart and Sharir [135] relates Davenport-Schinzel sequences and the combinatorial structure of lower envelopes of collections of functions; this relationship plays an important role to bound the complexity of some kinetic data structures, discussed in Section 4.1.2.

Theorem 11 (Hart and Sharir [135]). *Let $\mathcal{W}$ be a collection of n partially-defined, continuous, univariate functions, with at most s intersection points between graphs of any pair. Then the length of the lower envelope sequence of $\mathcal{W}$ is at most $\lambda_{s+2}(n)$.*

Figure 4.1 illustrates this relationship with a possible set of line segments and its corresponding Davenport-Schinzel sequence with $s = 3 \rightarrow 1, 2, 1, 3, 1, 3, 2, 4, 5, 4, 5, 2, 3$.

4.1.2   Kinetic Data Structures

To represent continuous motions on a given geometric data structure, one could associate a function of time to each one of its primitives, which are usually points on the *Euclidean space*; such functions are called *trajectories*. Functions applied on moving points are

---

[1] Any known finite quantity q in the universe gives $\alpha(q) \leq 5$.
[2] Nivasch was attributed the best student paper awards of SODA'09 for this work.
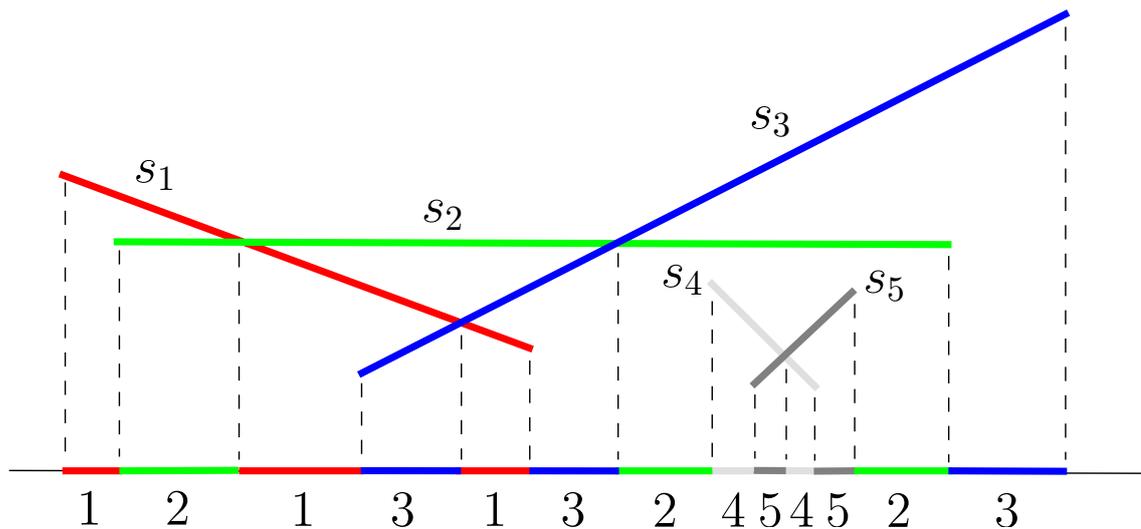
Figure 4.1: **Lower envelope of some line segments.** *Segments s_1, s_2, s_3, s_4, s_5 give the Davenport-Schinzel sequence:* $1, 2, 1, 3, 1, 3, 2, 4, 5, 4, 5, 2, 3$.

consequently a function of time. Namely, when the points move along a trajectory, certificates become *certificate functions* of time. The domain of these trajectories is usually an interval contained in $\mathbb{R}$, which is denoted by *timeline*.

Without loss of generality, a given certificate is valid while its corresponding certificate function is non-negative. For continuous trajectories, if a given geometric data structure is valid, then it is certified to be valid until at least the value of one of its certificate functions changes sign; this happens, when the current time in the timeline corresponds to one root of a certificate function. Such changes trigger an *event*, and an event might trigger (sometime expensive) updates in the data structure. Updates consist basically in one or several *discrete changes*; the number of such discrete changes is related to the number of events, which in turn is related to the complexity of the combinatorial structure of lower envelopes of collections of functions.[3] Geometric data structures maintained under this motion model, which was introduced by Basch et al. in 1997 [40], are called *kinetic data structures*. Note that one certificate failure does not necessarily imply an invalid geometric data structure, unless the chosen set of certificates is *minimal*.

In the last fifteen years, the scientific community designed, analyzed, and implemented a variety of kinetic data structures: e.g., kinetic maximum [11], kinetic sorted order [11, 167], kinetic convex hull [40, 22, 12], kinetic closest pair [40], kinetic collision detection [211, 149, 39, 10], kinetic triangulations [15, 19, 145], kinetic Delaunay triangulations in two and three dimensions [20, 187], and kinetic regular triangulations in two and three dimensions [133, 187].

Primitives are allowed to arbitrarily change their trajectory at any time, as long as they remain continuous. Such events are called *trajectory update*. Whenever a trajectory update occurs, all the certificates involved must be re-solved in order to guarantee the validity of the structure. We may simplify the continuous trajectories of the primitives to polynomials, thus each coordinate could be represented by a polynomial of time. Then, certificate functions are also polynomials of time, and their roots can be handled exactly.

---

[3]The Theorem 11 bounds the complexity of the combinatorial structure of such lower envelopes.

General Algorithm. Assuming: (i) primitives are point in $\mathbb{R}^d$, and (ii) motions are defined between two time stamps t_a, t_b, such that $-\infty < t_a \leq t_b < \infty$; for any set $\mathcal{E}$ of certificate functions C_i, the kinetic data structure algorithm (detailed e.g., in Guibas [129]) can be schematized as follows:

- Input moving points $p(t)$ as trajectory functions defined on the interval $[t_a, t_b]$.

- For any certificate function $C_i \in \mathcal{E}$ evaluated on the input, compute a set of discrete certificate failure times. In other words, the roots r_{ij} of the certificate function C_i.

- Insert each root $r_{ij} \in [t_a, t_b]$ in a priority queue associated to an event. The head of the priority queue corresponds to the smallest root inserted.

- Take the first root in the priority queue. Fetch the associated event. Handle it by updating the geometric data structure, then removing the root from the priority queue. And so generate new certificate functions.

- When there are no more events, then the algorithm ends.

We distinguish two types of event: *internal* and *external*. *External events* are events associated with necessary changes in the configuration of the data structure. *Internal events*, on the other hand, are events where some certificate fails, but the overall desired configuration still remains valid (they are essentially an overhead). Roughly speaking, internal events are dummy events introduced by some certificate; the cause is often the choice of the set of certificates, which is not *minimal*.

For a given kinetic data structure finding a *good* set of certificates is a challenge: A good set of certificates assures the validity of the data structure and is inexpensive to maintain.

In general, the scientific community is concerned in providing *good* kinetic data structures. Four classical measures to evaluate whether a kinetic data structure is good are:

- *Locality:* how many certificates depend on each primitive.

- *Responsiveness:* worst case amount of work required to process a single event.

- *Efficiency:* the total cost of processing all events over some period.

- *Compactness:* the total number of certificates needed by the kinetic data structure.

Kinetic data structures performing well on those measures are said to be: local, responsive, efficient, and compact. Basch et al. [41] are the first to our knowledge to formalize these measures, but recently a stronger version of the formalization has been proposed by Alexandron et al. [22]:

- **Locality:** A kinetic data structure is *local*, if the maximum number of events at any fixed time in the queue that are associated with a particular primitive is no more than polylogarithmic in the number of input primitives.

- **Responsiveness:** A kinetic data structure is *responsive*, if the processing time of an event by the repair mechanism is no more than polylogarithmic in the number of input primitives.

- **Efficiency:** A kinetic data structure is *efficient*, if the ratio between the number of internal events to the number of external events is no more than polylogarithmic in the number of input primitives.

- **Compactness:** A kinetic data structure is *compact*, if the space used by the data structure is larger than the number of input primitives by at most a polylogarithmic factor.

Next, we present some of the kinetic triangulations maintained under this motion model. The focus is, amongst the four criterion discussed above, the **efficiency** criterion.

4.1.3   Kinetic Delaunay Triangulations

Among all triangulations of $\mathbb{R}^d$, one of the most interesting is the Delaunay triangulation under the Euclidean metric. The most efficient kinetic data structure known up to 2010, to our knowledge, is due to Albers et al. [20], which dates back from 1998.[4] This result is summarized in Theorem 12.

Theorem 12 (Albers et al [20]). *Given a finite set $S(t)$ of n moving points describing polynomial trajectories of bounded degree in $\mathbb{R}^d$, the maximum number of events[5] over time is $O(n^d \lambda_s(n))$, where s is a constant depending only on the trajectories.*

In the plane,[6] whether $O(n)$ *Steiner points*[7] are allowed to be added or not, the best lower bound up to 2010 is quadratic [16]. Their construction works for any triangulation in the plane, and hence being a very general result. Theorem 13 summarizes this result.

Theorem 13 (Agarwal et al. [16]). *There exists a scenario of n points moving in the plane at constant speed so that any kinetic triangulation of linear size, with possibly a linear number of Steiner points, requires $\Omega(n^2)$ discrete changes over the course of motion.*

As we can see, there is a huge gap between the quadratic lower bound and the near cubic upper bound (in the plane). It is strongly believed that Delaunay triangulations of moving points under the Euclidean metric in the plane have a near-quadratic maximum number of discrete changes. And, not only experimental evidences corroborate with this belief, but also theoretical evidences: Delaunay triangulations of moving points (with constant velocity) under the L^1 or L^∞ metric produce $O(n^2 \alpha(n))$ discrete changes [71]. Actually, closing this gap is one of the most important open problems in computational geometry at the present moment [90].[8]

---

[4]The original idea is even older, and comes from Guibas et al. [131] for Delaunay triangulations of moving points in the plane.

[5]Such events in the plane lead to an expected $O(1)$ discrete changes.

[6]Most of the theoretical results related to triangulations in this area until 2010, are about triangulations in the plane.

[7]Additional dummy points intended to improve complexity.

[8]Progress have been recently done in this direction: In the 26th Symposium on Computational Geometry, Agarwal et al. [17] introduced a subgraph of the Delaunay graph in the plane that undergoes a nearly quadratic number of discrete changes.

4.1.4   Kinetic Triangulations

Notorious works have been done to find expected local, responsive, efficient and compact kinetic triangulations in the plane. The journey starts with: (i) a naive $O(n^4)$ kinetic triangulation; then passes through (ii) a $O(n^3\beta_s(n))$ kinetic Delaunay triangulation (see Section 4.1.3); through (iii) Agarwal et al.'s $O(n^{7/3})$ kinetic triangulation [15]; through (iv) Agarwal et al.'s $n^2 2^{O(\sqrt{\log n \log \log n})}$ hierarchical kinetic triangulation [19]; and finally ends with (v) a near-quadratic $\tilde{O}(n^2)$ [9] kinetic triangulation [145] presented in the 26th Symposium on Computational Geometry. (The journey for a near-quadratic kinetic Delaunay triangulation in the plane keeps on.) We briefly describe three important landmarks of this journey: *kinetic fan triangulations* [15], *kinetic hierarchical fan triangulations* [19], and *kinetic treap-based triangulations* [145]. This section is largely borrowed from these three nice works [15, 19, 145]. (Kinetic Delaunay triangulations in L_1, L_∞, and polygonal metrics [71, 72] were known to be near-quadratic before kinetic treap-based triangulations; however formal measures on kinetic data structures [41] were not yet defined at this time, and only the efficiency criterion was guaranteed.)

Fan triangulation. Let $S = \{p_1, \ldots, p_n\}$ be a set of n (stationary) points in $\mathbb{R}^2$, sorted in non-increasing order of their y-coordinates. The fan triangulation of S is constructed by sweeping a horizontal line h from $y = +\infty$ to $y = -\infty$. At any time the algorithm maintains the fan triangulation of points from S that lie above h. It updates the triangulation when the sweep line crosses a point $p_i \in S$ by adding the edges $p_i p_j$ whenever p_i sees p_j; look at Figure 4.2. The triangulation at the end of the sweep is the fan triangulation of S.
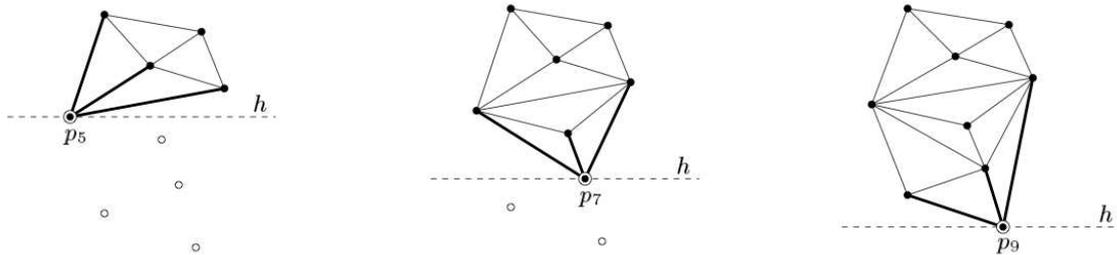
Figure 4.2: **Fan triangulation.** *Construction of fan triangulation at various stages — the points denoted by double circle is being inserted, and the thick edges are added (picture taken from Agarwal et al. [19]).*

Kinetic fan triangulations have two distinct types of event:

- **Ordering event.** It happens when two points are switching orders; i.e., exactly when two points in S have the same y-coordinates. See Figure 4.3(a).

- **Visibility event.** It happens when a point is becoming (in)visible; i.e., exactly when two adjacent edges of the fan triangulation become collinear. See Figure 4.3(b).

Theorem 14 bounds the number of discrete changes for kinetic fan triangulations and points in linear motion.

---

[9] $g = \tilde{O}(f(n))$ means that $g = O(f(n) \cdot h(n))$, where $h(n)$ is a polynomial of the logarithm of n.

(a)

(b)
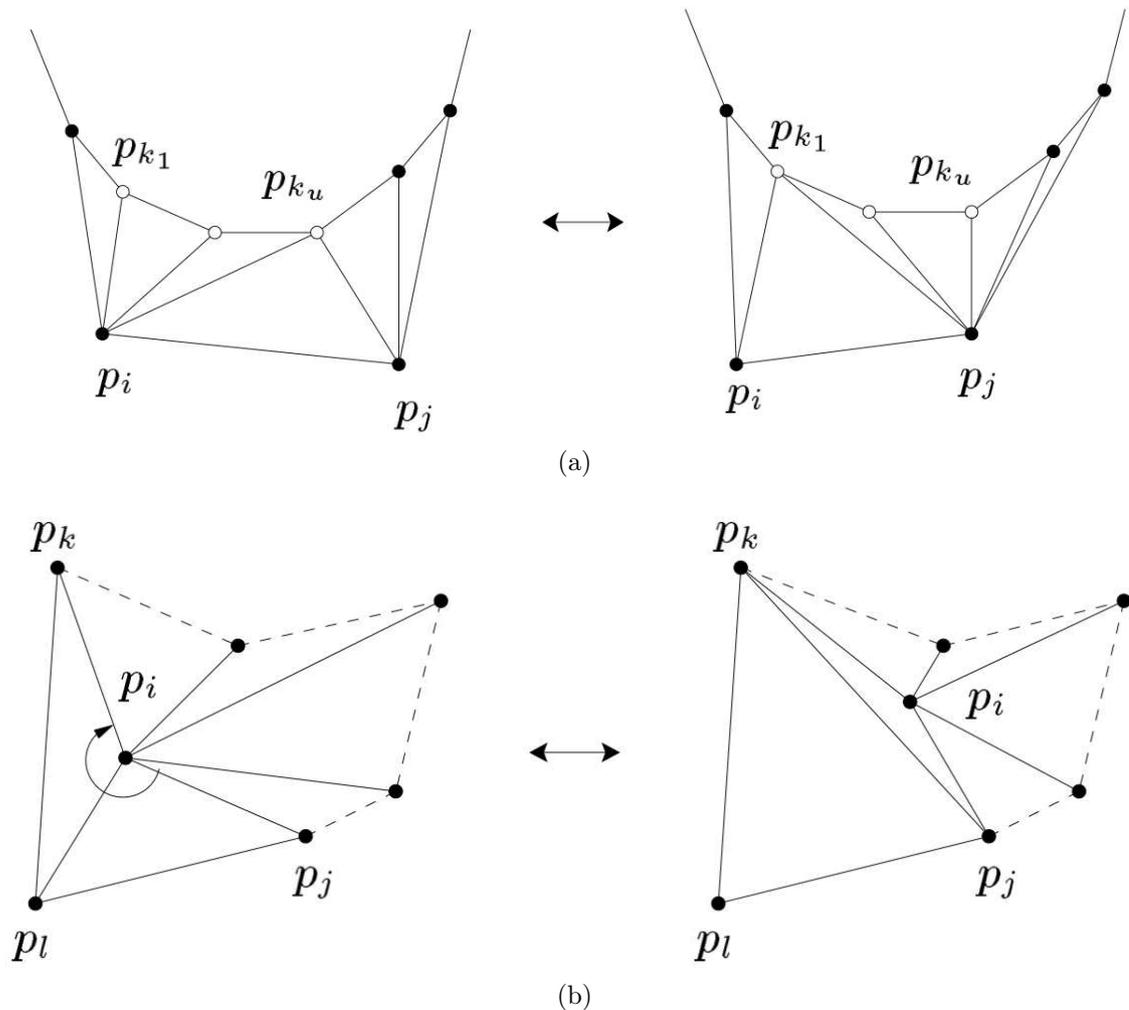
Figure 4.3: **Fan triangulation events.** *(a) ordering event, p_i and p_j switch order; (b) visibility event, $p_i p_j$ and $p_i p_k$ become collinear and finally p_j sees p_k. (Picture taken from Agarwal et al. [19].)*

Theorem 14 (Agarwal et al. [15]). *For points in linear motion, the number of discrete changes to the fan triangulation is bounded by $O(n^{4/3}\lambda_s(n))$, where s is a constant.*

Constrained fan triangulation. Later, Agarwal et al. introduced the *constrained fan triangulations* [19] as a tool to design more efficient kinetic triangulations. Let Λ be a set of segments with pariwise-disjoint interiors whose endpoints lie in S. The constrained fan triangulation of S and Λ is built almost in the same way as the fan triangulation of S. The only difference is that, during the sweep-line process, points cannot see each other when a segment of Λ blocks their vision; see Figure 4.4.

A kinetic constrained fan triangulation has another type of event:

- **Crossing event.** It happens when a point is crossing a segment of Λ; i.e., exactly when a point is on a segment of Λ.

Hierarchical fan triangulation. Based on constrained fan triangulations, Agarwal et al. defines *hierarchical fan triangulations*. Given a set of n points S in the plane,
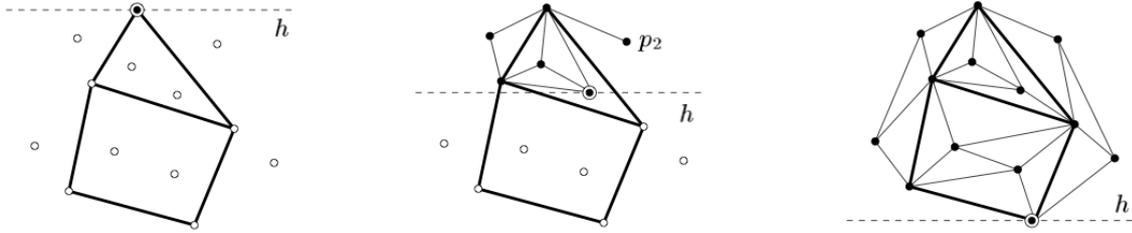
Figure 4.4: **Constrained fan triangulation.** *Constructing constrained fan triangulation with respect to Λ (thick edges) at various stages (picture taken from Agarwal et al. [19]).*

the hierarchical fan triangulation [19] constructs random samples $\emptyset = R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots \subseteq R_h = S$, and maintain a set of constrained fan triangulations F_i of S_i and Λ_i, for $i = 1, \ldots, h$, where $S_i = R_i$ and $\Lambda_i = F_{i-1}$; see Figure 4.5.
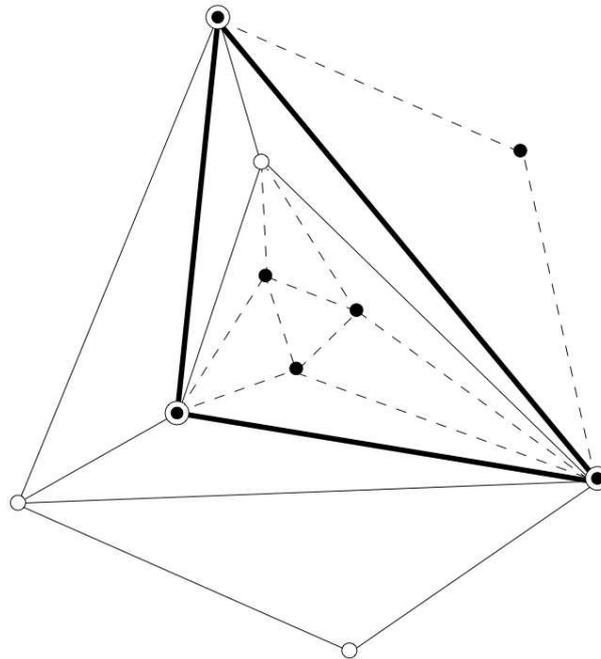
Figure 4.5: **Hierarchical fan triangulation.** *A hierarchical fan triangulation with three levels — points in the first level are denoted by double circles, second level by hollow circles, and third level by black circles (picture taken from Agarwal et al. [19]).*

A kinetic hierarchical fan triangulation has another type of event:

- **Hereditary event.** It happens when a topological change in F_i propagates changes in F_j, for $j > i$, as the insertion and deletion of an edge in F_i affects the visibility of points in R_j.

If we let $h = \left\lceil \sqrt{\log n / \log \log n} \right\rceil$ be the number of levels of the kinetic hierarchical fan triangulation, then we have $o(n^{2+\epsilon})$ events; this is summarized in Theorem 15.

Theorem 15 (Agarwal et al. [19])**.** *Let S be a set of n points moving in $\mathbb{R}^2$. If the motion of S is algebraic, a triangulation of S can be maintained that processes $n^2 2^{O(\sqrt{\log n \log \log n})}$ events, and each event requires $O(\log n)$ time to be processed.*

Note however that this bound is still not $\tilde{O}(n^2)$. And thus, using the measures in Section 4.1.2, the kinetic hierarchical fan triangulation is not yet efficient. The kinetic triangulation described in the sequel, is the first one, to our knowledge, to meet formally all the criteria.

Treap-based triangulation. Kaplan et al. [145] defines the *treap-based triangulation*[10] as follows: Given a set of points S, assign to each point of S a random *priority*, and sort them by their x-coordinates. Then, split S at the point p with the highest priority into a left portion S_L and a right portion S_R, compute recursively the upper convex hulls of $S_L \cup \{p\}$ and of $S_R \cup \{p\}$, and merge them into the upper convex hull of the whole set S. This process results into a *pseudo-triangulation* [186] of the portion of the convex hull of S lying above the x-monotone polygonal chain $\mathcal{C}(S)$ connecting the points S in their x-order. Each pseudo-triangle is x-monotone, and consists of an upper *base* and of a left and right lower concave chains, meeting at its bottom *apex*; Figure 4.6 depicts the anatomy of such a pseudo-triangle, whereas Figure 4.7 illustrates a global view of the recursive steps. A symmetric process is applied to the portion of the hull bellow $\mathcal{C}(S)$, by computing recursively lower convex hulls of the respective subsets of S.

The treap-based triangulation of S is obtained by partitioning each pseudo-triangle τ into triangles; this is accomplished by: (i) processing each vertex of τ in order, according to the random priority that they received, and (ii) drawing from each processed vertex v a chord, within the current sub-pseudo-triangle τ' of τ containing v, which split τ' into two sub-pseudo-triangles. This process ends with a triangulation of τ. The *treap* part of the **treap**-based triangulation's name is due to the fact that the recursive upper-hulls (or lower-hull) can be represented as a binary search tree on the x-coordinates of the points, and a heap with respect to the points priorities [145], i.e., a *treap* [28].
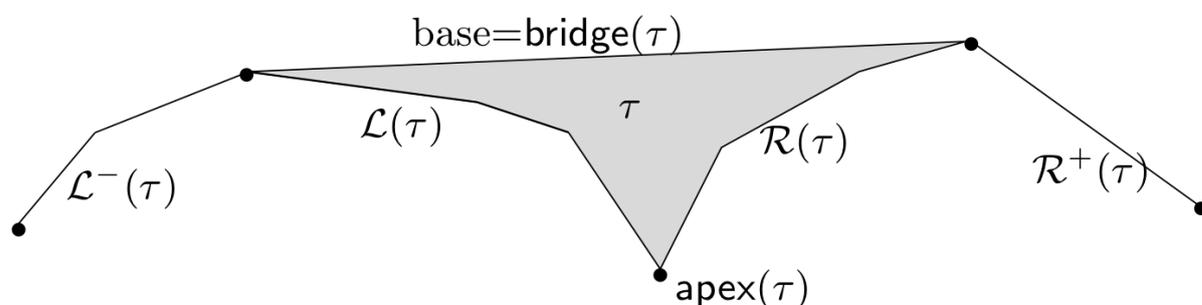
Figure 4.6: **Pseudo-triangle of the treap-based triangulation.** *Elements of a pseudo-triangle τ and the corresponding upper-hull (picture taken from Kaplan et al. [145]).*

Kinetic treap-based triangulations have three distinct type of event:

- **Envelope event.** It happens when a vertex, which is not an endpoint of bridge(τ), is being added to or removed from one of the chains bounding τ; i.e. when one of the chains $\mathcal{L}(\tau)$, $\mathcal{R}(\tau)$ contains three collinear vertices; see Figure 4.8(a).

- **Visibility event.** It happens when a point of $\mathcal{R}(\tau)$ and a point of $\mathcal{L}(\tau) \cup \mathcal{L}^-(\tau)$ are not seeing each other anymore, or vice-versa; see Figure 4.8(b).

---

[10]Kaplan et al. [145] did not give a name to their triangulation. The name we suggest here is neither original from Kaplan et al. [145] nor definitive.
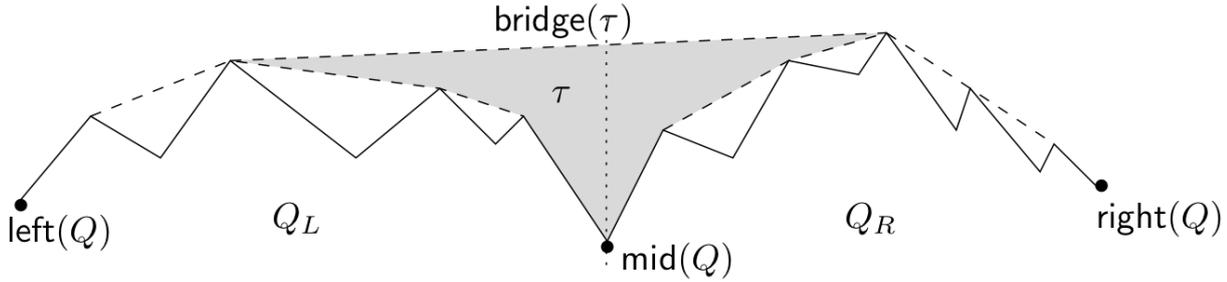
Figure 4.7: **Recursive pseudo-triangulation.** *Recursing on the left and right part of the pseudo-triangle τ (picture taken from Kaplan et al. [145]).*

- **Swap event.** It happens when some point p not belonging to τ and with higher priority than the (i) apex, (ii) upper-left point or (iii) upper-right point of τ, is crossing one of the vertical lines through these points; see Figure 4.8(c).
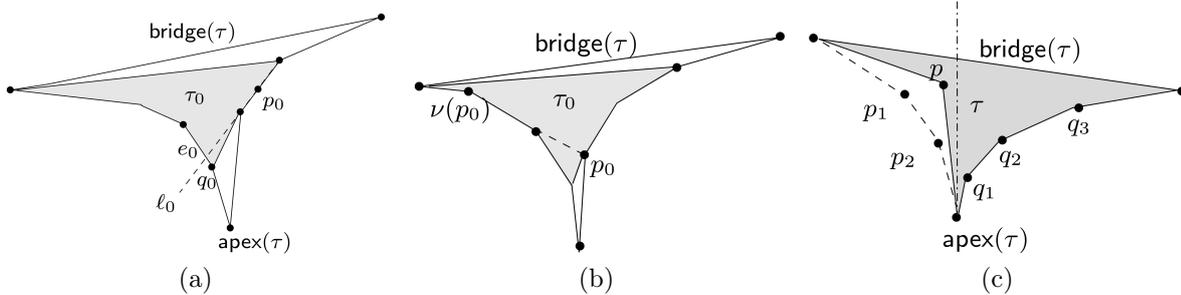
Figure 4.8: **Kinetic treap-based triangulation events.** *(a) visibility event; (b) envelope event. The sub-pseudo-triangle τ_0 contains all edges which are inserted to or deleted from the resulting triangulation of τ. (c) swap event. The funnel of τ immediately before the x-swap between p and the apex of τ, which causes the vertices p_1 and p_2 to appear on $\mathcal{L}(\tau)$, and the vertices q_1, q_2, q_3 to disappear from $\mathcal{R}(\tau)$. (Picture taken from Kaplan et al. [145].)*

Kaplan et al. [145] proved that the kinetic treap-based triangulation processes an $\tilde{O}(n^2)$ number of events, which is the first local, responsive, efficient, and compact kinetic triangulation (see Section 4.1.2); the efficiency requirement is summarized in Theorem 16.

Theorem 16 (Kaplan et al. [145]). *Let S be a set of n points moving in $\mathbb{R}^2$. A triangulation of S can be maintained that processes an expected number of $O(n^2 \beta_{s+2}(n) \log n)$ events, each in $O(\log^2 n)$ expected time, where s is the maximum number of times at which any single triple of points in S may become collinear.*

At this point, we reiterate that bounds on the number of discrete changes discussed above are expected in the random order of the indices, and not in the worst case.

4.2 Time Stamp Relocation Context

Several applications, such as mesh smoothing, mesh optimization, re-meshing, and some physical simulations to name a few, can be modeled in terms of time stamp relocations.

For these applications, standard kinetic data structures tend to be overwhelming in practice: On one hand, events occurring between two time stamps in a standard kinetic data structure are often useless in a time stamp relocation context; and on the other hand, the granularity of two time stamps are often high enough so that triangulation snapshots at two consecutive time stamps are "reasonably" similar.

Within the time stamp relocation context, a simple method consists in rebuilding the whole triangulation from scratch at every time stamp. We denote such an approach by **_rebuilding_**. When samples of the output at each timestamp have an expected linear size, rebuilding can lead to an expected $O(kn \log(n))$ time complexity [93], where n denotes the number of vertices of the triangulation and k denotes the number of distinct time stamps. Despite its "poor" theoretical complexity, the rebuilding algorithm turns out to be surprisingly hard to outperform when most of the points move, as already observed [187]. Furthermore, rebuilding the triangulation from scratch allows us to use the most efficient static algorithms: Delaunay triangulation construction from Cgal [223, 179] is an example of such efficient static algorithms; it sorts the points so as to best preserve point proximity for efficient localization, and make use of randomized incremental constructions.

There are a number of applications, which require computing the next vertex locations one by one, updating the Delaunay triangulation after each relocation [24, 216, 217]. Naturally, rebuilding is unsuitable for such applications, since it requires the knowledge of all the vertex locations at once. Another naive updating algorithm, significantly different from rebuilding, is the **_relocation_** algorithm, which relocates the vertices one by one. Roughly speaking, the latter consists of iterating over all vertices to be relocated; for each relocated vertex the algorithm first walks through the triangulation to locate the simplex containing its new position, inserts a vertex at the new position and removes the old vertex from the triangulation. And thus, each relocation requires three operations for each relocated point: (i) one point location, (ii) one insertion, and (iii) one deletion. When the displacement of a moving point is small enough the point location operation is usually fast. In favorable configurations with small displacement and constant local triangulation complexity, the localization, insertion, and deletion operations take constant time per point. This leads to $O(m)$ complexity per time stamp, where m is the number of moving points. Such complexity is theoretically better than the $O(n \log n)$ complexity of rebuilding. In practice however, the deletion operation is very costly and hence rebuilding the whole triangulation is faster when all vertices are relocated; i.e., when $m = n$.

We recall from Section 2.2.3 that: (i) algorithms which are not able to relocate vertices one by one are referred to as static; (ii) algorithms relocating vertices one by one are referred to as dynamic. Advantages of being dynamic include to name a few:

(i)   the computational complexity depends mostly on the number of moving points (which impacts on applications where points eventually stop moving);

(ii) the new location of a moving point can be computed on the fly, which is required for variational methods [87, 24, 216, 217];

(iii) the references to the memory that the user may have, remain valid (this is not true for the rebuilding algorithm).

Rebuilding is static while the relocation algorithm is dynamic.

Several recent approaches have been proposed to outperform the two naive algorithms (rebuilding and relocation) in specific circumstances. For example, *kinetic data structures* [129] are applicable with a careful choice of vertex trajectories [187]. Some work has also been done to improve the way kinetic data structures handle degeneracies [12], when the timeline can be subdivided in time stamps. Approaches based on *kinetic data structures* are often dynamic as they can relocate one vertex at a time. Guibas and Russel consider another approach [128], which consists of the following sequence: (i) remove some points until the triangulation has a non-overlapping embedding; then (ii) *flip* [112, 54] the invalid pairs of adjacent cells until the triangulation is valid (i.e., Delaunay); and finally (iii) insert back the previously removed points. In this approach, the flipping step may not work in dimensions higher than two [200]; when that happens rebuilding is triggered, and rebuilds the triangulation from scratch with a huge computational overhead. For their input data however, such *stuck cases* do not happen too often, and rebuilding can be outperformed when considering some heuristics on the order of points to remove. Although this method can be adopted in a dynamic way, when relocating one vertex at a time, it loses considerably its efficiency as it is not allowed to use any heuristic anymore. Shewchuk proposes two elegant algorithms to repair Delaunay triangulations: *Star Splaying* and *Star Flipping* [204]. Both algorithms can be used when *flipping* do not work, instead of rebuilding the triangulation from scratch. Finally, for applications that can live with triangulations that are not necessarily Delaunay at every time stamp (e.g., almost-Delaunay triangulations upon lazy deletions [87]), some dynamic approaches outperform dynamic relocation by a factor of three [87]. It is worth mentioning that dynamic algorithms performing nearly as fast as rebuilding are also very well-suited for applications based on variational methods.

When dynamicity is not required, rebuilding the triangulation at each time stamp is definitely an option, and performs well in practice. We show here some attempts of the scientific community to outperform the rebuilding approach on the last years:[11] (i) kinetic data structures within a time stamp relocation context (Section 4.2.1); (ii) static methods (Section 4.2.2); and (iii) almost-Delaunay structures (Section 4.2.3).

4.2.1   Kinetic Data Structures

Kinetic data structures can be used to update Delaunay triangulations within a time stamp relocation context as well [187]. Let us first, remind a few points that should be considered when looking at kinetic data structures: (i) certificate functions, (ii) event processing, (iii) event generation, and (iv) trajectories.

A valid triangulation in two or three dimensions, with a single extra vertex at infinity connected to each hull facet [48], can be checked to be Delaunay using the empty-sphere certificate; the definition of an empty-sphere certificate and its details along with the vertex at infinity and derived notions can all be found in Section 2.2.

- **Two Dimensions.** Given a sequence of four points $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $p_3(x_3, y_3)$, $p_4(x_4, y_4)$ of a pair of adjacent cells (also denoted by *faces* in 2D, when there is no ambiguity), the empty-sphere certificate (also denoted by *empty-circle* certificate in

---

[11]There are some very recent progress in the field, including techniques derived from this work's contribution in Chapter 5 [224].

2D) function equation is the determinant of the following matrix:

$$\begin{pmatrix} x_1(t) & y_1(t) & x_1(t)^2 + y_1(t)^2 & 1 \\ x_2(t) & y_2(t) & x_2(t)^2 + y_2(t)^2 & 1 \\ x_3(t) & y_3(t) & x_3(t)^2 + y_3(t)^2 & 1 \\ x_4(t) & y_4(t) & x_4(t)^2 + y_4(t)^2 & 1 \end{pmatrix}, \tag{4.2}$$

and, given a sequence of three finite points $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $p_3(x_3, y_3)$ of a pair of infinite adjacent cells (also convex hull's *edges* in 2D), the empty-circle function becomes the determinant of the following matrix:

$$\begin{pmatrix} x_1(t) & y_1(t) & 1 \\ x_2(t) & y_2(t) & 1 \\ x_3(t) & y_3(t) & 1 \end{pmatrix}. \tag{4.3}$$

- **Three Dimensions.** The situation is analogous here. Given a sequence of five points $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$, $p_3(x_3, y_3, z_3)$, $p_4(x_4, y_4, z_4)$, $p_5(x_5, y_5, z_5)$ of a pair of adjacent cells, the empty-sphere certificate function equation is the determinant of the following matrix:

$$\begin{pmatrix} x_1(t) & y_1(t) & z_1(t) & x_1(t)^2 + y_1(t)^2 + z_1(t)^2 & 1 \\ x_2(t) & y_2(t) & z_2(t) & x_2(t)^2 + y_2(t)^2 + z_2(t)^2 & 1 \\ x_3(t) & y_3(t) & z_3(t) & x_3(t)^2 + y_3(t)^2 + z_3(t)^2 & 1 \\ x_4(t) & y_4(t) & z_4(t) & x_4(t)^2 + y_4(t)^2 + z_4(t)^2 & 1 \\ x_5(t) & y_5(t) & z_5(t) & x_5(t)^2 + y_5(t)^2 + z_5(t)^2 & 1 \end{pmatrix}, \tag{4.4}$$

and, given a sequence of four finite points $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$, $p_3(x_3, y_3, z_3)$, $p_4(x_4, y_4, z_4)$ of a pair of infinite adjacent cells, the empty-sphere certificate function becomes the determinant of the following matrix:

$$\begin{pmatrix} x_1(t) & y_1(t) & z_1(t) & 1 \\ x_2(t) & y_2(t) & z_2(t) & 1 \\ x_3(t) & y_3(t) & z_3(t) & 1 \\ x_4(t) & y_4(t) & z_4(t) & 1 \end{pmatrix}. \tag{4.5}$$

In two dimensions, the intersection of a pair of adjacent faces such that the empty-circle certificate fails is an edge; such an edge is called a *bad edge*. Analogously, in three dimensions, a *bad facet* is a facet that is the intersection of a pair of adjacent cells such that the empty-sphere certificate fails. For Delaunay triangulations, an event being processed means updating its combinatorial structure when required; i.e., repairing its simplices, by getting rid of the bad edges (or the bad facets in 3D).

In two dimensions, for an embedded triangulation, it is well-known that flipping the bad edges successively suffices to correctly update the entire combinatorial structure [205]; i.e., make the triangulation Delaunay. Thus, in two dimensions an event can be processed by simply flipping the bad edge and generating four new certificates each time [187, 20]. However, in three dimensions, it is not known whether flipping is enough to update the combinatorial structure in this kinetic scenario. (If points do not move continuously, flipping is not enough [144].) It is common in three dimensions processing events by

vertex insertion and deletion; Russel's experiments [187] however, indicates that flipping works much of the time, and produces six near empty-sphere certificates each time.

Recall that, in the time stamp relocation context, we are considering that points move discretely at a finite sequence of time stamps. This should be contrasted with kinetic data structures, where points move continuously. To model a relocation time stamp context based on a kinetic Delaunay triangulation, for each interval between two time stamps t_a and t_b, we need to choose a trajectory for each moving point $p(t)$ of the input set: Such a trajectory can be any function of time, as long as (i) it is continuous, (ii) starts at $p(t_a)$, and (iii) ends at point $p(t_b)$. Next, we look at some interesting choices for the trajectories.

A natural choice for the trajectory is curves parametrized by a polynomial of the time; such a choice leads to certificate functions that are also polynomials of time, and their roots can be extracted with some of the available exact algebraic equation solvers [188]. Taking a look at Eq.(4.2) and Eq.(4.4), we notice that using arbitrary polynomials as interpolant may lead to certificate functions with high degrees. Since, at some point, we compute roots of certificate functions, some strategy has to be considered to avoid searching for roots of high-degree polynomials.

Several strategies were proposed by Russel in his PhD Thesis [187]. We present them here as a small survey. Let $T(d)$ be the time needed for finding roots on a polynomial of degree d, f the number of cells in the initial triangulation and e_η the number of events which occur when the points are moved to their final position using the strategy η.

- *Linear interpolation (LI).* Pick a constant velocity motion for each moving point starting at its initial position and ending at its final position. In other words, allow x, y, z to change linearly at the same time. The resulting certificates have degree four in two dimensions and five in three dimensions. The cost to maintain the structure is then $(f + 6e_{LI})T(5)$ in three dimensions.

- *Lifted Linear interpolation (LLI).* Please, have a look at Eq.(4.4), $l = x_i{}^2 + y_i{}^2 + z_i{}^2$ is the coordinate of (x_i, y_i, z_i) on the *lifted space* [114]. If the kinetic data structure can also handle the added complexity of being a kinetic regular triangulation [36], which is the case of Russel's [187], then the coordinate on the lifted space can be interpolated as well, and thus decreasing the degree of the polynomial by one. The cost to maintain the structure becomes $(f + 6e_{LLI})T(4)$ in three dimensions.

- *Point at a time (PAT).* If we manage to vary each row on Eq.(4.4) one at a time, we get linear polynomials as certificate functions. It corresponds to moving each point as in *LLI*, but one after another. Since each certificate must be recomputed five times in three dimensions, one for each point involved, then the cost becomes $(5f + 6e_{PAT})T(1)$.

- *Coordinate at a time (CAT).* The same idea above can be applied for the coordinates, by moving each coordinate one at a time. The total number of trajectory changes becomes four (coordinate l included). Therefore, the cost becomes $(4f + 6e_{CAT})T(1)$.

Russel evaluates the performance of these strategies in several data sets. The data sets, tables and benchmarks in the rest of this section can be found in Russel's original work [187] with more details.

- From **molecular simulations**, *hairpin* and *protein A*: A short 177 atom beta hairpin and Staphyloccocal protein A, a 601 atom globular protein.

- From **muscle simulations**, *bicep*: A volumetric data from a simulation of a bicep contracting. The points move comparatively little between the frames.

- From **falling object simulations**, *falling objects* and *shifting objects*: Those sets are taken from a simulation of a collection of 1000 objects dropped into a pile. Initially, the objects fall through the air with occasional contacts, but later in the simulation they form a dense, although still shifting pile. The two data sets are related with those two distinct phases of the simulation, and are called *falling objects* and *shifting objects* respectively.

Further details on those data sets are shown in Table 4.1.

name	points	cells	ephemeral cells	empty-sphere tests	tol.	20% tol.
hairpin	177	1114	2554	7516	8.5	1.5
protein A	601	3943	10250	31430	8.6	1.5
bicep	3438	21376	66553	210039	9.0	1.6
falling objects	1000	6320	17137	52958	12	1.7
shifting objects	1000	6381	17742	55299	13	1.2

Table 4.1: **Attributes of the static Delaunay triangulation.** *Ephemeral cells* are *cells created during the construction process which are not in the final triangulation. There were generally three time as many ephemeral cells as cells in the final triangulation. Their number gives some idea of the excess work done by the rebuilding process.* **Tol.** *is the average fraction of the local edge length that a point must move so as to invalidate an empty-sphere certificate. The 20%* **tol.** *is the fraction of the local average edge length that points need to move to invalidate 20% of the certificates. Very small displacements compared to the edge length can invalidate cells in a Delaunay triangulation.*

Naively implemented, kinetic data structures cannot compete with rebuilding, even when ignoring exactness issues as it is shown in Table 4.2. The running time is dominated by the initialization; i.e., generating the costs and solving the certificate function for each pair of adjacent cells of the initial triangulation.

Nevertheless, the fastest strategy was the *coordinate at a time*, with *point at a time* being almost as fast; the latter actually caused fewer events in general, however its extra initialization cost made it slightly slower. Both techniques were about a factor of two more expensive than rebuilding the triangulation.

As mentioned before, even adopting any of the aforementioned strategies, the naive approach of kinetic data structure performance is far behind rebuilding's. One way to reduce the cost of maintaining the combinatorial structure, is to adopt some filtering techniques. Four layers of filtering were proposed by Russel [187]. Naturally, if none of the filters succeed, the sign of the roots of the certificate function are computed exactly, using an exact polynomial equation solver [188].

- *Layer 0* (avoiding activation) tries to avoid redundancies by allowing a small set of vertices to "jump" directly to their final position when no certificate would be invalidated.

motion	hairpin	protein A	bicep	falling objects	shifting objects
linear	.04	.03	.04	.06	.06
lifted linear	.07	.08	.09	.10	.07
coordinate at a time	.50	.62	.39	.48	.54
point at a time	.40	.30	.40	.51	.44

Table 4.2: **Kinetic Delaunay update costs.** *Speedups compared to rebuilding and event counts for different interpolating motions are shown. Note that all algorithms are slower than rebuilding and so all the costs are smaller than one. The coordinate at a time based methods are within a factor of two of rebuilding.*

- *Layer 1* (interval arithmetic) first evaluates the empty-sphere certificate functions at the current time stamp with interval arithmetic [178, 56], then if 0 is included on the output interval, evaluates them exactly; this is similar to the arithmetic filtering presented in Section 2.1.2.

- *Layer 2* (using the derivative) uses a lower bound on the derivative of the certificate function in an attempt to prove that such certificate function does not fail until the next time stamp.

- *Layer 3* (approximate the failure time) uses Newton's approximation method to produce an approximation of when a certificate function vanishes; i.e., it finds the intervals where the roots of the certificate function are contained: the *root intervals*. If successful, Layer 3 can prove that the certificate function has no roots in the interval over which it is being solved, or find an interval that contains the first root.

Table 4.3 indicates the filter failure ratio achieved by each filtering layer.

data set	filter 0	filter 1	filter 2	filter 3
good	4%	12%	54%	0%
borderline	68%	29%	6%	2%

Table 4.3: **Filter failures.** *The table shows the failure rates for each level of filters. On the borderline data, there were many more failures of the early filters, which were caught by the derivative filter in level two (after the certificate function was generated). The inputs are divided into **good** and **borderline** sets. The former are data sets where the filters (especially filter 0 and 1) perform well and the update procedure works effectively, the latter where they do not work as well.*

After mixing all those ingredients together, the kinetic data structure approach may be faster than rebuilding, but only when there is far less than 1% bad certificates overall; this strongly indicates that kinetic data structure is not well-suited within a time stamp relocation context.

Acar et al. [12] improve the way kinetic data structure works for convex hull in three dimensions,[12] mostly in the case of several triggers of close events in time. They divide

---

[12]Convex hulls in three dimensions relate to Delaunay triangulations in two dimensions [57].

time into a lattice of fixed-size intervals, and process events at the resolution of an interval; see Figure 4.9. The original idea comes from [13] and the main motivation is to avoid ordering the roots of polynomials, which is slow when roots are close to each other (because of the precision required to compare them exactly).
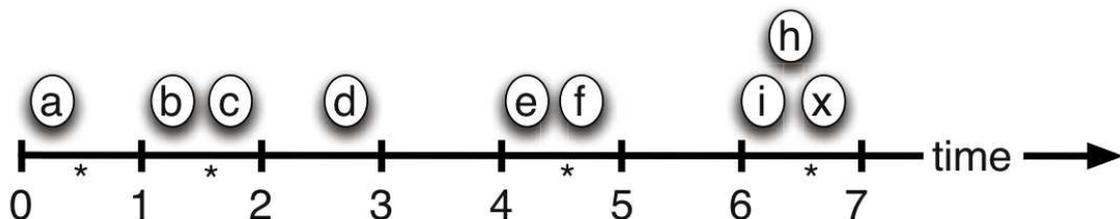
Figure 4.9: **Dividing time into a lattice.** *The lattice and events by time interval (picture taken from Acar et al. [12]).*

4.2.2   Static Techniques

When moving points from a Delaunay triangulation between two discrete time stamps, there is a family of static techniques that only involves computing certificates on the initial and final coordinates and directly transforming the initial triangulation into the final answer. Hence, those techniques are comparatively easier to make them run faster than their kinetic counterpart. Guibas and Russel [128] came up with a number of such techniques; they are presented next.

Let $(\mathcal{T}, \Phi)$ signify assigning the coordinates of a sequence of points $\Phi = \{p_i\}_{i=0}^n$ to the sequence of vertices $\Phi = \{v_i\}_{i=0}^n$ of the triangulation $\mathcal{T}$, and $\mathcal{DT}(\Phi)$ signify the Delaunay triangulation of the points in Φ.[13] Thus, for two sequences of points S and S', the combinatorial structures of the Delaunay triangulation of S and S' are equal when $(\mathcal{DT}(S), S')$ is also Delaunay; the definition of combinatorial structure can be found in Section 2.2.1. To verify whether $(\mathcal{DT}(S), S')$ is Delaunay: (i) check whether all cells are correctly oriented, then (ii) check whether all pair of adjacent cells have a positive evaluation of the empty-sphere certificate. Whenever $(\mathcal{DT}(S), S')$ is not Delaunay, one could remove from S a subsequence of points Ψ one by one, in such a way that $(\mathcal{DT}(S \backslash \Psi), S')$ is Delaunay. Then reinsert Ψ on the triangulation. Since $(\mathcal{DT}(\emptyset), S')$ is trivially Delaunay, removing $\Psi = S$ suffices to guarantee the correctness of this technique. This kind of static technique is called *point deletion*. In order to improve the performance of such a technique, points could have some sort of score to evaluate the deletion order. A total of three distinct heuristics to compute the score of points were considered by Guibas and Russel [128], they are:

- *random.* Each vertex is given a score of one if it is involved in an invalid certificate, and zero otherwise; this means picking any points that is involved in an invalid certificate.

---

[13]For short, in this discussion, any Delaunay triangulation of a sequence of points is assumed to be unique, which is not true in general; however such an issue can be handled by using the *symbolic perturbation* method [99] in order to make the Delaunay triangulation unique for any input sequence.

- *worst first.* Each vertex is assigned a score based on how many failed certificates it is involved in.

- *geometry.* Each vertex is assigned a score based on how much the displacement has locally distorted the geometry of the point set. Such a heuristic is likely to use more expensive computations; however it uses not only combinatorial properties of the sets S and S', but geometrical properties as well.

On the aforementioned process, vertex deletion is likely to be called several times on each iteration; vertex deletion [70, 94, 99] is expensive in both two and three dimensions (it is much more expensive than insertion). In two dimensions, Chew's randomized vertex deletion[14] algorithm [70] is a nice option. In three dimensions, one of the most pro-eminent method so far is Devillers and Teillaud's [99], which the robustness is guaranteed by the *symbolic perturbation* method (implemented in CGAL [179]).

Besides, whenever points are removed and placed anywhere else, many new cells and facets are created, leading to a significant quantity of certificates generated. On the other hand, flips can be much cheaper and easier to analyze because of the constant number of certificates generated: four for two dimensions and six for three dimensions. If the triangulation is embedded, then, in two dimensions, any triangulation can be made Delaunay with such flips. In three dimensions, it is not guaranteed; whenever it is **not** possible to made a three-dimensional triangulation Delaunay by successive flips, we call such a case a *stuck case*. In most of the three-dimensional triangulations that Guibas and Russel [128] experimented, they could made them Delaunay with successive flips.

With this in mind, an alternative algorithm consists in: (i) removing a set of points Ψ from S, such that $(\mathcal{DT}(S\backslash\Psi), S')$ is embedded (defined in Section 2.2.4); then (ii) trying to use successive flips to finish the conversion to Delaunay. For two-dimensional triangulations, this algorithm works well and fast. Conversely, for three-dimensional triangulations, removing points can no longer be safely applied in the stuck cases. Moreover, after the displacements, the triangulation is not necessarily Delaunay anymore and the hole created by removing a point from a non-Delaunay triangulation in three dimensions may not be able to be filled with cells [200]. An alternative for the stuck cases consists in: (i) finding a point incident to a non-Delaunay cell that can be removed from the triangulation, then (ii) removing it. If even that alternative fails, then the method recomputes the triangulation from scratch. This static technique is called *hybrid*, and works well in both two and three dimensions, because, in practice, the stuck cases seem to be rare enough [128].

Benchmarks of those static methods using data sets detailed in Table 4.1 come from the original work of Russel [187], where more details can be found. From those benchmarks, amongst the three heuristics above, the *worst first* heuristic achieved the best running time, followed closely by the *random* heuristic. In general, they outperformed rebuilding when 1-2% of the cells were invalidated by vertex displacements. The fastest method overall was the *hybrid* method which could outperform rebuilding when even around 10% of the cells were invalidated by vertex displacements. Nevertheless, the frequency of stuck cases was less than 10% per time stamp in average for their data sets. Benchmarks are detailed in Table 4.4; Guibas and Russel's attempt to use geometry as a heuristic was unsuccessful [187], and results are omitted.

---

[14]Chew's algorithm builds the Delaunay triangulation of vertices of a convex polygon, which is equivalent to vertex deletion in a two-dimensional Delaunay triangulation.

simulation	step	random	worst first	hybrid
hairpin	1	.65	.65	**1.4**
protein A	1	.32	.42	**1.5**
	2	.06	.24	**1.1**
bicep	1	**1.8**	**2.0**	**3.1**
	2	**1.3**	**1.3**	**2.7**
	4	.70	.78	**2.3**
	8	.08	.10	.73
falling objects	1	**1.4**	**1.6**	**3.5**
	2	**1.2**	.93	**3.5**
	4	.38	.53	**2.0**
	8	.81	.22	**1.4**
shifting objects	1	.46	.60	**2.2**
	2	.42	.44	**1.2**

Table 4.4: **Static update performance.** *The table shows the speedups to patch the triangulation compared to rebuilding and the number of deletions for the presented heuristics. Entries in **boldface** are ones which are better than rebuilding (i.e., larger than one).*

Guibas and Russel achieve in their experiments a speed-up factor of about three times on rebuilding for some input data. Moreover, *Star Splaying* and *Star Flipping* [204] can be used when *flipping* causes a deadlock, instead of rebuilding the triangulation from scratch. Since, if the points don't move to *far*, Star Splaying is linear with respect to the number of vertices, then if the constants hidden in the big-O notation is not too large, it has some potential to outperform rebuilding in practice; however, to the best of our knowledge, no implementation of these methods is currently available.

4.2.3 *Almost*-Delaunay Structures.

Not every application needs to maintain a Delaunay triangulation at each single time stamp, but only some "reasonable" triangulation instead. In that case, it may be relevant to find some significantly cheaper almost-Delaunay scheme. This is specially useful for surface reconstruction and re-meshing problems: DeCougny and Shephard [86] use an almost-Delaunay property based on an empty circumsphere criterion in order to accelerate their surface re-meshing scheme; Debard et al. [87] use an almost-Delaunay property based on lazy operations to lessen the computation time of their 3D morphing application.

We believe that the work described in [87] illustrates well how the cost of maintaining deforming surfaces can be reduced with such almost-Delaunay structures. Their morphing algorithm aims to minimize some energy function, and works as follows:

(1) sample a set of points S from the surface;

(2) discretize the surface by triangulating S (ideally a Delaunay triangulation);

(3) minimize an energy function (e.g., the Gaussian energy [220]) on the cells of the triangulation.

Since the third step is hard, Debard et al. use a similar scheme as Meyer et al.'s [166]: The scheme consists of a simple gradient descent method, where only one point moves at a time. It is exactly for that task that their algorithm would benefit from a faster Delaunay triangulation update for moving points. Debard et al. [87] propose a dynamic scheme to trade quality of mesh for speed of computation, by relaxing the Delaunay condition and moving points one at a time. Naturally, there is a trade-off between convergence speed and computation time. The Delaunay condition is relaxed by the means of lazy updates, which associates a score to each vertex, indicating how strongly its position would violate the embedding and Delaunay properties after a displacement:

- If the vertex displacement would make a triangulation invalid, its score is incremented by s_1.

- If the vertex displacement would make a triangulation non-Delaunay, its score is incremented by s_2.

When the score of a vertex reaches s_{limit}, the vertex is relocated. Experimentally, they have founded that $(s_1, s_2, s_{limit}) = (2, 1, 4)$ is a good trade-off between *Delaunayness* of the structure and convergence speed. See [87] for further details.

Their benchmark consists of moving three thousand points from a random initial distribution of points to a well-shaped ellipsoid. The experiment reflects the relative cost of each certificate on the average: (i) relocating vertices naively, and (ii) using their almost-Delaunay structure. As shown in Table 4.5, the largest portion of the running-time is due to the deletion of vertices in both cases. However, the alternative method could save around half the time due to deletion operations. Still, in both cases, moving points is the bottleneck of their reconstruction algorithm.

algorithm	predicate	CPU time (%)
relocation		92.59
	remove	74.31
	empty-sphere	6.58
	insert	2.90
	orientation	0.62
	locate	0.06
internal		7.41
almost-Delauany		81.15
	remove	49.00
	empty-sphere	11.59
	insert	2.70
	orientation	2.06
	locate	0.07
internal		18.85

Table 4.5: Benchmarks, taken from the work of Debard et al. [87].

Debard et al. achieve a three time factor speed-up on the naive relocation algorithm. Their scheme is above twice slower than rebuilding, however it is dynamic.

Chapter 5

A Filtering Scheme for Point Relocations on Delaunay Triangulations

---

"To those who ask what the infinitely small quantity in mathematics is, we answer that it is actually zero. Hence there are not so many mysteries hidden in this concept as they are usually believed to be." — Leonhard Euler.

- P. M. M. de Castro, J. Tournois, P. Alliez and O. Devillers. Filtering relocations on a Delaunay triangulation. In *Computer Graphics Forum, 28*, pages 1465–1474, 2009. Note: Special issue for EUROGRAPHICS Symposium on Geometry Processing. (Also available as: Research Report 6750, INRIA, 2008.)

- P. M. M. de Castro and O. Devillers. Fast Delaunay Triangulation for Converging Point Relocation Sequences. In *Abstracts 25th. European Workshop on Computational Geometry*, pages 231-234, 2009.

For several applications, points move at some discrete time stamps. And at each time stamp, it is required to maintain the Delaunay triangulation of those points. This happens for instance in data clustering [136, 30], mesh generation [87], re-meshing [218, 23], mesh smoothing [25], mesh optimization [24, 216, 67], to name a few. Moreover, this operation is often the bottleneck in these applications.

Two naive solutions exist: *rebuilding* and *relocation*. Several other solutions exist and are detailed in Section 4.2.

When all the points move, rebuilding is a very good option in practice. However, when points move with a small magnitude, or when only a fraction of the vertices move, rebuilding is no longer the best option. This chapter contributes with a solution specially designed for these cases.

Our Contributions. We propose to compute for each vertex of the triangulation a safety zone where the vertex can move without changing its connectivity. This way each relocation that does not change the connectivity of the triangulation is *filtered*. We

show experimentally that this approach is worthwhile for applications where the points are moving under small perturbations.

Our main contribution takes the form of a dynamic filtering method that maintains, within a time stamp relocation context, a Delaunay triangulation of points relocating with small amplitude. The noticeable advantages of the filter are: (i) **Simplicity.** The implementation is simple as it relies on well-known dynamic Delaunay triangulation constructions [93, 100] with few additional geometric computations; and (ii) **Efficiency.** Our filtering approach outperforms in our experiments by at least a factor of four, in two and three dimensions, the current dynamic relocation approach used in mesh optimization [216]. It also outperforms the rebuilding algorithm for several conducted experiments. This opens new perspectives, e.g, in mesh optimization, the number of iterations is shown to impact the mesh quality under converging schemes. The proposed algorithm enables the possibility of going further on the number of iterations.

In Section 5.1, we introduce two relevant regions associated with a vertex of a Delaunay triangulation: (i) the safe region and (ii) the tolerance region of a vertex. Also, we delineate the relationship between these regions and the connectivity of the triangulation. Then, in Section 5.2, we present Algorithm 1, which is a filtering algorithm based on the tolerance region, and the main contribution of this chapter. In Section 5.3, we obtain some constants for certified vertex tolerance computations. And finally, in Section 5.4, we present several experiments in both two and three dimensions in order to evaluate how Algorithm 1 behaves in practice.

5.1   Defining New Regions

We now introduce the notions of *safe region* and *tolerance region* of a vertex.

5.1.1   Tolerances

Let the function $C : \mathcal{Z}^m \rightarrow \{-1, 0, 1\}$ be a certificate acting on a m-tuple of points $\zeta = (p_1, p_2, \ldots, p_m) \in \mathcal{Z}^m$, where $\mathcal{Z}$ is the space where the points lie. By abuse of notation, $p \in \zeta$ means that p is one of the points of ζ, and ζ_{p_i} is the i-th point in ζ. We define the *tolerance* of ζ with respect to C, namely $\epsilon_C(\zeta)$ or simply $\epsilon(\zeta)$ when there is no ambiguity, the largest displacement applicable to $p \in \zeta$ without invalidating C. More precisely, the tolerance, assuming $C(\zeta) > 0$, can be stated as follows:

$$\epsilon_C(\zeta) = \inf_{\substack{\zeta' \\ C(\zeta') \leq 0}} \max_{i=1\ldots m} \|\zeta_{p_i} - \zeta'_{p_i}\|, \tag{5.1}$$

where $\|.\|$ is the Euclidean norm of a vector.

Let $\mathcal{X}$ be a finite set of m-tuples of points in $\mathcal{Z}^m$. Then, the *tolerance* of an element $\mathbf{e}$ belonging to one or several m-tuples of $\mathcal{X}$ with respect to a given certificate C and to $\mathcal{X}$, is denoted by $\epsilon_{C,\mathcal{X}}(\mathbf{e})$ (or simply by $\epsilon(\mathbf{e})$ when there is no ambiguity) and is defined as follows:

$$\epsilon_{C,\mathcal{X}}(\mathbf{e}) = \inf_{\substack{\zeta \ni \mathbf{e} \\ \zeta \in \mathcal{X}}} \epsilon_C(\zeta). \tag{5.2}$$

5.1.2 Tolerance in a Delaunay Triangulation

Call a pair of adjacent cells a *bi-cell*, then the tolerance involved in a Delaunay triangulation is the *tolerance of the empty-sphere certificate* acting on any bi-cell of a Delaunay triangulation; see Figure 5.1(a). From Eq.(5.1), it corresponds to the size of the smallest perturbation the bi-cell's vertices can undergo so as to become cospherical. This is equivalent to compute the hypersphere that minimizes the maximum distance to the $d + 2$ vertices; i.e., the *optimal middle sphere*, which is the median sphere of the d-annulus of minimum width containing the vertices; see Figures 5.1(d) and 5.1(e). An annulus, defined as the region between two concentric hyperspheres, is a fundamental object in computational geometry [122].

Dealing with Boundary. When dealing with infinite bi-cells, namely the ones which include the point at infinity, the *d-annulus of minimum width* becomes the region between two parallel hyperplanes. This happens because the hypersphere passing through ∞ reduces to an hyperplane.

In order to precise which are those parallel hyperplanes, we consider two distinct cases (that coincide in two dimensions):

- If only one cell of the bi-cell is infinite then one hyperplane H_1 passes through the d common vertices of the two cells. The other hyperplane H_2 is parallel to H_1 and passes through the unique vertex of the finite cell which is not contained in the infinite cell; see Figure 5.2(a).

- Otherwise the two vertices which are not shared by each cell form a line L and the remaining finite vertices are a ridge of the convex hull H (an edge in three dimensions). Consider the hyperplanes: H_1 passing through H and parallel to L; and H_2 passing through L and parallel to H. They compose the boundary of the d-annulus; see Figure 5.2(b).

5.1.3 Tolerance Regions

Let $\mathcal{T}$ be a triangulation lying in $\mathbb{R}^d$ and $\mathcal{B}$ a bi-cell in $\mathcal{T}$; see Figure 5.1(a). The *interior facet* of $\mathcal{B}$ is the common facet of both cells of $\mathcal{B}$; see Figure 5.1(b). The *opposite vertices* of $\mathcal{B}$ are the remaining two vertices that do not belong to its interior facet; see Figure 5.1(c). We associate to each bi-cell $\mathcal{B}$ an arbitrary hypersphere $\mathcal{S}$ denoted by *delimiter* of $\mathcal{B}$; see Figure 5.1(e). If the interior facet and opposite vertices of $\mathcal{B}$ are respectively inside and outside the delimiter of $\mathcal{B}$, we say that $\mathcal{B}$ verifies the *safety condition*; we call $\mathcal{B}$ a *safe bi-cell*. If a vertex p belongs to the interior facet of $\mathcal{B}$, then the *safe region* of p with respect to $\mathcal{B}$ is the region inside the delimiter; otherwise, the *safe region* of p with respect to $\mathcal{B}$ is the region outside the delimiter. The intersection of the safe regions of p with respect to each one of its adjacent bi-cells is called *safe region* of p. If all bi-cells of $\mathcal{T}$ are safe bi-cells we call $\mathcal{T}$ a *safe triangulation*. When a triangulation is a safe triangulation, we say that it verifies the *safety condition*.

It is clear that a safe triangulation implies a Delaunay triangulation as:

- Each delimiter can be shrunk so as to touch the vertices of the interior facet, and thus defines an empty-sphere passing through the interior facet of its bi-cell (which proves that the facets belongs to the Delaunay triangulation).
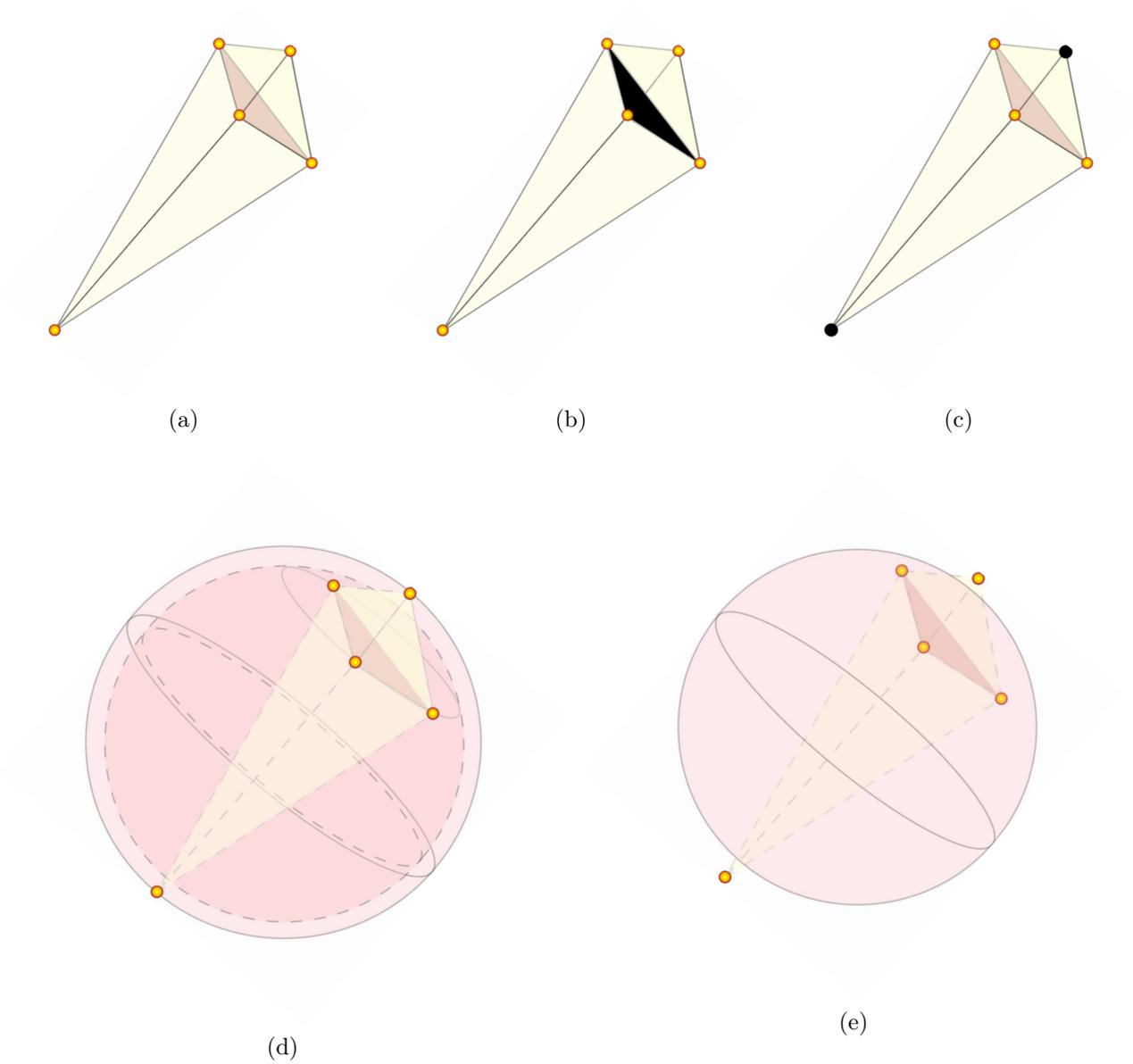
Figure 5.1: **Definitions.** *(a) A three-dimensional bi-cell, (b) its interior facet and (c) opposite vertices. (d) The boundaries of its 3-annulus of minimum width; the smallest boundary passing through the facet vertices and the biggest boundary passing through the opposite vertices. (e) depicts its standard delimiter separating the facet and opposite vertices.*

Then we have the following proposition (the definition of combinatorial structure can be found in Section 2.2.1):

Proposition 17. Given the combinatorics of a Delaunay triangulation $\mathcal{T}$, if its vertices move inside their safe regions, then the triangulation obtained while keeping the same combinatorics as in $\mathcal{T}$ in the new embedding remains a Delaunay triangulation.

Proposition 17 is a direct consequence of the implication between safe and Delaunay triangulations: If the vertices remain inside their safe regions then $\mathcal{T}$ remains a safe triangulation. As a consequence it remains a Delaunay triangulation.
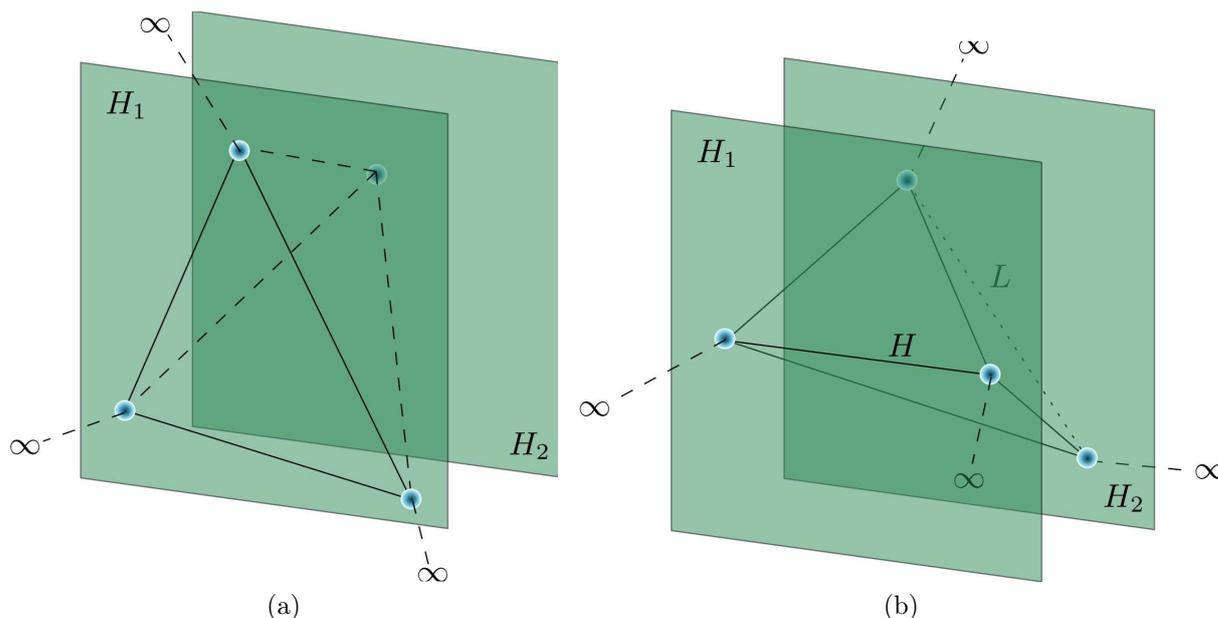
Figure 5.2: **Infinite bi-cells.** *The* 3-*annulus of minimum width of a bi-cell containing: (a) one infinite cell and one finite cell, (b) two infinite cells.*

Note that the safe region of a vertex depends on the choice of the delimiters for each bi-cell of the triangulation. We denote by *tolerance region* of p, given a choice of delimiters, the biggest ball centered at the location of p included in its safe region. More precisely, let $\mathcal{D}(\mathcal{B})$ be the delimiter of a given bi-cell $\mathcal{B}$. Then, for a given vertex $p \in \mathcal{T}$, the tolerance region of p is given by:

$$\tilde{\epsilon}(p) = \inf_{\substack{\mathcal{B} \ni p \\ \mathcal{B} \in \mathcal{T}}} dist(p, \mathcal{D}(\mathcal{B})), \tag{5.3}$$

where $dist$ is the Euclidean distance between a point and a hypersphere. We have $\tilde{\epsilon}(p) \leq \epsilon(p)$, since the delimiter generated by the minimum-width d-annulus of the vertices of a bi-cell $\mathcal{B}$ maximizes the minimum distance of the vertices to the delimiter; see Figure 5.3.

Among all possible delimiters of a bi-cell, we define the *standard delimiter* as the median hypersphere of the d-annulus with the inner-hypersphere passing through the interior facet and the outer-hypersphere passing through the opposite vertices. Both median hypersphere and d-annulus are unique. We call the d-annulus the *standard annulus*. If our choice of delimiter for each bi-cell of $\mathcal{T}$ is the standard delimiter, then we have $\tilde{\epsilon}(p) = \epsilon(p)$. Notice that the standard annulus is usually the annulus of minimum width described in Section 5.1.2 defined by the vertices of $\mathcal{B}$. In pathological cases where the minimum-width annulus is not the standard annulus, then the standard delimiter is not safe [122].

Computing the standard annulus of a given bi-cell $\mathcal{B}$ requires computing the center of a d-annulus. This is achieved through finding the line perpendicular to the interior facet passing through its circumcenter (which corresponds to the intersection of the bisectors of the vertices of the interior facet of $\mathcal{B}$) and intersecting it with the bisector of the opposite vertices of $\mathcal{B}$.
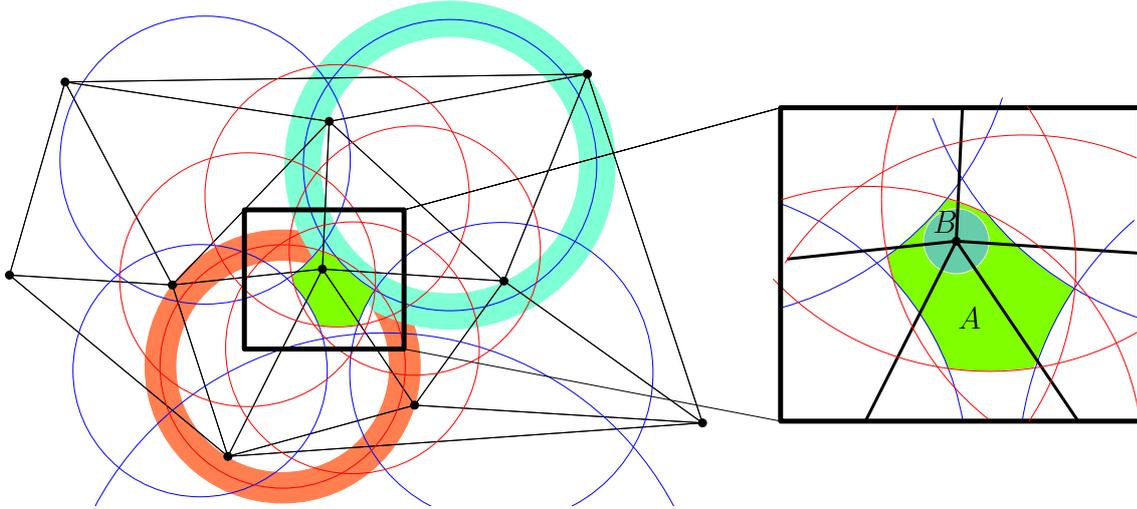
Figure 5.3: **Safe region and tolerance region.** $p \in \mathbb{R}^2$ *the center of* B. *The region* A *is the safe region of p, while* B *is its tolerance region.*

5.2   Filtering Relocations

The dynamic relocation algorithm, which relocates one vertex after another, unlike rebuilding, can be used when there is no *a priori* knowledge about the new point locations of the whole set of vertices. Dynamic approaches are especially useful for algorithms based on variational methods, such as [87, 24, 216, 217].

In this section, we propose two improvements over the naive relocation algorithm for the case of small displacements. Finally, we detail two algorithms devised to filter relocations using the tolerance and safe region of a vertex respectively in a Delaunay triangulation (see Section 5.1.3). The former being fast and of practical use.

5.2.1   Improving the Relocation Algorithm for Small Displacements

In two dimensions, a modification of the relocation algorithm leads to a substantial acceleration: by a factor of two in our experiments. This modification consists of flipping edges when a vertex displacement does not invert the orientation of any of its adjacent triangles. The key idea is to avoid as many deletion operations as possible, as they are the most expensive. In three dimensions, repairing the triangulation is far more involved [204].

A weaker version of this improvement consists of using the relocation algorithm only when at least one topological modification is needed; otherwise the vertex coordinates are simply updated. Naturally, this additional computation leads to an overhead, though our experiments show that it pays off when displacements are small enough. When this optimization is combined with the first algorithm described in the next section, our experiments show evidence that it is definitely a good option.

5.2.2   Filtering Algorithms

We now redesign the relocation algorithm so as to take into account the tolerance region of every relocated vertex. The *filtering algorithms* proposed in this section are capable

of correctly deciding whether or not a vertex displacement requires an update of the connectivity so as to trigger the trivial update condition. They are dynamic in the sense that points can move one by one, like the naive relocation algorithm.

Vertex Tolerance Filtering

- **Data structure.** Consider a triangulation $\mathcal{T}$, where for each vertex $v \in \mathcal{T}$ we associate two point locations: f_v and m_v. We denote them respectively by the *fixed* and the *moving* position of a vertex. The fixed position is used to *fix* a reference position for a moving point; whereas the moving position of a given vertex is its actual position, and changes at every relocation. Initially, the fixed and moving positions are equal. We denote by $\mathcal{T}_f$ and $\mathcal{T}_m$ the embedding of $\mathcal{T}$ with respect to f_v and m_v respectively. For each vertex, we store two numbers: ϵ_v and D_v;[1] they represent respectively the tolerance value of v and the distance between f_v and m_v.

- **Pre-computations.** We initially compute the Delaunay triangulation $\mathcal{T}$ of the initial set of points S, and for each vertex we set $\epsilon_v = \epsilon(v)$ and $D_v = 0$. For a given triangulation $\mathcal{T}$, the tolerance of each vertex is computed efficiently by: (i) computing tolerances of all the bi-cells in $\mathcal{T}$, and (ii) keeping the minimum value on each vertex.

- **Computations.** For every vertex displacement, run Algorithm 1.

- **Correctness.** Algorithm 1 is shown to terminate as each processed vertex v: either (i) gets a new displacement value $D_v \leq \epsilon_v$ when the filter succeeds (in line 1); or (ii) gets a new displacement value $D_v = 0 \leq \epsilon_v$ when the filter fails (in line 8). In such a situation, from Proposition 17, $\mathcal{T}_m$ is a Delaunay triangulation.

- **Complexity.** The tolerance algorithm has the same complexity as the relocation algorithm, given that relocation orders do not alter the complexity of the relocation algorithm. If all points move, then the amortized number of calls to the relocation algorithm per vertex is at most 2; this is due to the fact that when a vertex is relocated with the relocation algorithm, its D value is necessarily set to 0 (in line 8), and thus it can't be relocated more than twice: one time due to its displacement, and a second time due to some other vertex displacement.

Safety Region Filtering

A natural idea is to replace the tolerance test by a more involved test that checks whether a vertex stays within its safe region. We could also run this *safety test* in case of failure of the first test. The modified algorithm can be describe as follows.

- **Data structure.** Consider a triangulation $\mathcal{T}$, where for each vertex $v \in \mathcal{T}$ we associate two points: f_v and m_v, as in Algorithm 1. For each bi-cell, we associate a point $c_{\mathcal{B}}$ and a value $r_{\mathcal{B}} > 0$,[2] representing the center and the radius of its standard delimiter respectively.

---

[1] ϵ_v and D_v are computed and stored squared for a better performance.

[2] $r_{\mathcal{B}}$ is computed and stored squared for a better performance.

Algorithm 1 Vertex Tolerance Filtering.

Require: Triangulation $\mathcal{T}$ after pre-computations, a vertex v of $\mathcal{T}$ and its new location p
Ensure: $\mathcal{T}$ updated after the relocation of v
1: $(m_v, D_v) \leftarrow (p, \|\mathbf{f}_v, p\|)$
2: **if** $D_v < \epsilon_v$ **then**
3:      we are done
4: **else**
5:      insert v in a queue $\mathcal{Q}$
6:      **while** $\mathcal{Q}$ is not empty **do**
7:          remove h from the head of $\mathcal{Q}$
8:          $(f_h, \epsilon_h, D_h) \leftarrow (m_h, \infty, 0)$
9:          update $\mathcal{T}$ by relocating h with the relocation algorithm optimized for small displacements, as described in Section 5.2.1
10:          **for** every new created bi-cell $\mathcal{B}$ **do**
11:              $\epsilon' \leftarrow$ half the width of the standard annulus of $\mathcal{B}$
12:              **for** every vertex $w \in \mathcal{B}$ **do**
13:                  **if** $\epsilon_w > \epsilon'$ **then**
14:                      $\epsilon_w \leftarrow \epsilon'$
15:                      **if** $\epsilon_w < D_w$ **then**
16:                          insert w into $\mathcal{Q}$
17:                      **end if**
18:                  **end if**
19:              **end for**
20:          **end for**
21:      **end while**
22: **end if**

- **Pre-computations.** Compute the Delaunay triangulation $\mathcal{T}$ of the initial set of points, and for each bi-cell compute its standard delimiter.

- **Computations.** For every vertex displacement, run Algorithm 2.

- **Correctness.** Correctness is ensured by the same arguments used for the correctness of Algorithm 1.

- **Complexity.** Similarly to Algorithm 1, the amortized number of calls to the relocation algorithm per vertex is at most 2. However, the complexity of the safety test is proportional to the number of bi-cells incident to a vertex, which is in the worst case $O(n)$; and thus the worst-case complexity of Algorithm 2 is worse than Algorithm 1. However, in the average case, for several distributions, the number of bi-cells incident to a vertex is $O(1)$; and in these cases, Algorithm 1 and 2 have the same complexity.

Back to Vertex Tolerance Filtering

We can expect that while the tolerance algorithm takes a shorter time to verify whether a point m_v is inside its tolerance region, the safe region algorithm accepts (significantly)

Algorithm 2 Safety Region Filtering.

Require: Triangulation $\mathcal{T}$ after pre-computations, a vertex v of $\mathcal{T}$ and its new location p

Ensure: $\mathcal{T}$ updated after the relocation of v

1: $m_v \leftarrow p$
2: **if** m_v is inside the safe region of v **then**
3:     we are done
4: **else**
5:     insert v in a queue $\mathcal{Q}$
6:     **while** $\mathcal{Q}$ is not empty **do**
7:         remove h from the head of $\mathcal{Q}$
8:         $f_h \leftarrow m_h$
9:         update $\mathcal{T}$ by relocating h with the relocation algorithm optimized for small displacements, as described in Section 5.2.1
10:        **for** every new created bi-cell $\mathcal{B}$ **do**
11:            compute and store $c_{\mathcal{B}}$ and $r_{\mathcal{B}}$ with respect to $\mathcal{T}_f$
12:            **for** every vertex $w \in \mathcal{B}$ **do**
13:                **if** m_w belongs to the interior facet of $\mathcal{B}$ **then**
14:                    **if** $\|m_w, c_{\mathcal{B}}\| > r_{\mathcal{B}}$ **then**
15:                        insert w into $\mathcal{Q}$
16:                    **end if**
17:                **else**
18:                    **if** $\|m_w, c_{\mathcal{B}}\| < r_{\mathcal{B}}$ **then**
19:                        insert w into $\mathcal{Q}$
20:                    **end if**
21:                **end if**
22:            **end for**
23:        **end for**
24:    **end while**
25: **end if**

---

bigger vertex displacements without calling the relocation algorithm; compare regions A and B in Figure 5.3. However, the safety test is rather involved and does not save any computation time in practice, even when using first order approximations. In our experiments, Algorithm 2 is strictly worse than Algorithm 1, and we dropped it.[3]

When relocations of the vertices are governed by a convergent scheme, we can run rebuilding for the first few time stamps until the points are more or less stable, then switch to the filtering algorithm. As a drawback, the algorithm is no longer dynamic during the first time stamps. We give more details on this approach in Section 5.4.2.

Another point concerns robustness issues: Computing the tolerance values using floating-point computations may, in some special configurations, produce rounding errors, and hence wrong evaluation of ϵ_v. The algorithm ensures that T_f, the embedding at fixed position, is always correct while the embedding T_m at moving position may be incorrect. A certified correct Delaunay triangulation for T_m is obtained through a certified

---

[3]That bad news is adjourned in purpose; we believe that Algorithm 2 gives an illustration of a recurring phenomenon we experienced in this work: Some inviting ideas do not work in practice as smoothly as it seems to do.

lower bound computation of ϵ_v (see Section 5.3). As expected, the numerical stability of ϵ_v depends on the *quality of the simplices*. To explain this fact, consider the smallest angle θ between the line l perpendicular to the interior facet of a bi-cell passing through its circumcenter and the bisector h of the opposite vertices; see Figure 5.4. The numerical stability depends on the size of such an angle. For convex bi-cells, if θ is too small, the two simplices are said to have a *bad shape* [69]. It is worth saying that the computation of ϵ_p is rather stable in the applications that would benefit from filtering, since the shape of their simplices improves over time.
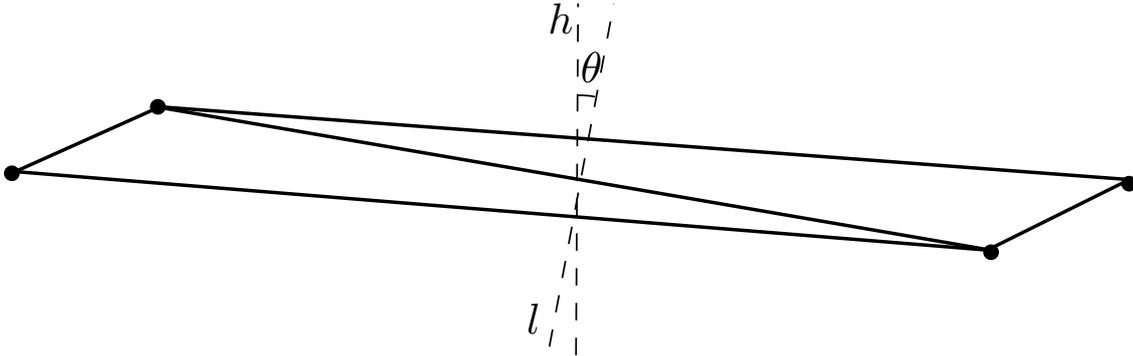
Figure 5.4: **Numerical stability.** *If the smallest angle θ between the $(d-2)$-dimension flat h and the line l is too small, the simplices are badly shaped in the sense of being non isotropic.*

5.3 Certified Computations

In this section, we show the tolerance of the empty-sphere certificate computations in two and three dimensions for finite bi-cells. We also show the details on the arithmetic filtering scheme in order to certify the computations; the notion of arithmetic filtering can be reviewed in Section 2.1.2. Assume: (i) the inputs are upper-bounded by a power of 2, represented by L; (ii) $\tau_{\{z_1,\ldots,z_k\}}$ and $\alpha_{\{z_1,\ldots,z_k\}}$ are a bound on the absolute value and error of the variables $z_1,\ldots,z_k$ respectively. Also, let us use the standard term $\mathrm{ulp}(x)$, and denote $\mu = \mathrm{ulp}(1)/2$. (The error bound formulas of $+,-,\times,/,\sqrt{}$ and the definition of $\mathrm{ulp}(x)$ are given in Section 2.1.3.)

```
/** Given a 2D bi-cell B with vertices
 *  v_i (x_i, y_i), i = 1, 2, 3, 4,
 *  compute the squared bi-cell tolerance.
 *  One circle passes through v_1, v_4;
 *  and the other through v_2, v_3
 */
1: squared_half_annulus_width(p_1, p_2, p_3, p_4)
2: {
3:    /* One bisector */
4:    a_1 := 2 * (x_2 - x_4);
5:    b_1 := 2 * (y_2 - y_4);
6:    c_1 := ((x_4)^2 + (y_4)^2) - ((x_2)^2 + (y_2)^2);
7:    /* The other bisector */
```

```
 8:    a_2 := 2 * (x_1 - x_3);
 9:    b_2 := 2 * (y_1 - y_3);
10:    c_2 := ((x_3)^2 + (y_3)^2) - ((x_1)^2 + (y_1)^2);
11:    /* The intersection between them */
12:    g := (a_1 * b_2) - (b_1 * a_2);
13:    n_x := (b_1 * c_2) - (b_2 * c_1);
14:    n_y := (a_2 * c_1) - (a_1 * c_2);
15:    /* The coordinates of the center of the annulus */
16:    den := 1 / g;
17:    x_annulus := n_x * den;
18:    y_annulus := n_y * den;
19:    /* The smallest annulus squared radius */
20:    r_2 := (x_2 - x_annulus)^2 + (y_2 - y_annulus)^2;
21:    /* The biggest annulus squared radius */
22:    R_2 := (x_3 - x_annulus)^2 + (y_3 - y_annulus)^2;
23:    /* The squared in-circle tolerance */
24:    w := (r_2 + R_2 - 2 * sqrt(r_2 * R_2))/4;
25:    return w;
26:}
```

These are error bounds for g, n_x, and n_y:

$$\alpha_g \leq 128L^2\mu,\ \alpha_{\{n_x,n_y\}} \leq 192L^3\mu.$$

Computations are detailed in Table 5.1. Since g is a divisor, we need to find an error bound of its inverse, represented by den, semi-statically with Eq.(2.3). Take $U = f(g) - \alpha_g$, then:

$$\alpha_{\mathtt{den}} \leq \frac{128L^2\mu}{U^2} + \left(\frac{1}{U}\right)\mu,\ \tau_{\mathtt{den}} \leq \frac{1}{U}.$$

From the seventeenth line on, pre-computed error bounds become too big (compared with the tolerances). We use the equations in Section 2.1.3 to dynamically evaluate a tighter upper bound of the absolute error. Note that bi-cells are from a triangulation which is supposedly Delaunay, and hence g cannot be 0.

variables	τ	α
$\{a_1, b_1, a_2, b_2\}$	$\leq 4L$	$\leq 4L\mu$
$\{c_1, c_2\}$	$\leq 4L^2$	$\leq 12L^2\mu$
$\{g\}$	$\leq 32L^2$	$\leq 4\alpha_{\{a_1,\dots\}}\tau_{\{a_1,\dots\}} + 4\tau^2_{\{a_1,\dots\}}\mu$ $\leq 64L^2\mu + 64L^2\mu$ $\leq 128L^2\mu$
$\{n_x, n_y\}$	$\leq 32L^3$	$\leq 2(\tau_{\{c_1,c_2\}}\alpha_{\{a_1,\dots\}} + \tau_{\{a_1,\dots\}}\alpha_{\{c_1,c_2\}}) + 4\tau_{\{a_1,\dots\}}\tau_{\{c_1,c_2\}}\mu$ $\leq 2(16L^3\mu + 48L^3\mu) + 64L^3\mu$ $\leq 192L^3\mu$

Table 5.1: **Certified 2D tolerance computation.** *Error bounds for the computation in 2D of the squared tolerance of a bi-cell.*

```
/** Given a 3D bi-cell B with vertices
 *  v_i (x_i, y_i, z_i), i = 1, 2, 3, 4, 5,
 *  compute the squared bi-cell tolerance.
 *  One sphere passes through v_1, v_5;
 *  and the other through v_2, v_3, v_4
 */
1: squared_half_annulus_width(v_1, v_2, v_3, v_4, v_5)
2: {
3:    double sa = (x_1)^2 + (y_1)^2 + (z_1)^2;
4:    double sb = (x_2)^2 + (y_2)^2 + (z_2)^2;
5:    double sc = (x_3)^2 + (y_3)^2 + (z_3)^2;
6:    double sd = (x_4)^2 + (y_4)^2 + (z_4)^2;
7:    double se = (x_5)^2 + (y_5)^2 + (z_5)^2;
8:    /* One bisector */
9:    double a_1 = 2*(x_1 - x_5);
10:   double b_1 = 2*(y_1 - y_5);
11:   double c_1 = 2*(z_1 - z_5);
12:   double d_1 = sa - se;
13:   /* The second bisector  */
14:   double a_2 = 2*(x_2 - x_4);
15:   double b_2 = 2*(y_2 - y_4);
16:   double c_2 = 2*(z_2 - z_4);
17:   double d_2 = sb - sd;
18:   /* The third bisector */
19:   double a_3 = 2*(x_4 - x_3);
20:   double b_3 = 2*(y_4 - y_3);
21:   double c_3 = 2*(z_4 - z_3);
22:   double d_3 = sd - sc;
23:   /* The intersection between them */
24:   double m01 = a_1*b_2 - a_2*b_1;
25:   double m02 = a_1*b_3 - a_3*b_1;
26:   double m12 = a_2*b_3 - a_3*b_2;
27:   double x01 = b_1*c_2 - b_2*c_1;
28:   double x02 = b_1*c_3 - b_3*c_1;
29:   double x12 = b_2*c_3 - b_3*c_2;
30:   double y01 = c_1*a_2 - c_2*a_1;
31:   double y02 = c_1*a_3 - c_3*a_1;
32:   double y12 = c_2*a_3 - c_3*a_2;
33:   double z01 = a_1*b_2 - a_2*b_1;
34:   double z02 = a_1*b_3 - a_3*b_1;
35:   double z12 = a_2*b_3 - a_3*b_2;
36:   double g = m01*c_3 - m02*c_2 + m12*c_1;
37:   double x012 = x01*d_3 - x02*d_2 + x12*d_1;
38:   double y012 = y01*d_3 - y02*d_2 + y12*d_1;
39:   double z012 = z01*d_3 - z02*d_2 + z12*d_1;
40:   double den = 1/g;
41:   /* The coordinates of the center of the annulus */
```

```
42:  double x_annulus = x012 * den;
43:  double y_annulus = y012 * den;
44:  double z_annulus = z012 * den;
45:  /* The smallest annulus squared radius */
46:  r_2 := (x_2 - x_annulus)^2 + (y_2 - y_annulus)^2
       + (z_2 - z_annulus)^2;
47:  /* The biggest annulus squared radius */
48:  R_2 := (x_1 - x_annulus)^2 + (y_1 - y_annulus)^2
       + (z_1 - z_annulus)^2;
49:  /* The squared in-sphere tolerance */
50:  w := (r_2 + R_2 - 2 * sqrt(r_2 * R_2))/4;
51:  return w;
52:}
```

These are error bounds for g, $x012$, $y012$, and $z012$:

$$\alpha_g \leq 384L^3\mu, \ \alpha_{\{x012,y012,z012\}} \leq 5,568L^4\mu.$$

Computations are detailed in Table 5.2. Again, since g is a divisor, we need to find an error bound of its inverse, represented by `den`, semi-statically with Eq.(2.3). Take $U = f(g) - \alpha_g$, then:

$$\alpha_{\mathtt{den}} \leq \frac{2,944L^3\mu}{U^2} + \left(\frac{1}{U}\right)\mu, \ \tau_{\mathtt{den}} \leq \frac{1}{U}.$$

From forty-second line on, pre-computed error bounds become too big (compared with the tolerances). We use the equations in Section 2.1.3 to dynamically evaluate a tighter upper bound of the absolute error. Note that, as in the two-dimensional case, g cannot be 0.

5.4   Experimental Results

This section investigates through several experiments the size of the vertex tolerances and the width of the standard annulus in two and three dimensions. We discuss the performance of the rebuilding, relocation and filtering algorithms on several data sets.

We run our experiments on a Pentium 4 at 2.5 GHz with 1GB of memory, running Linux (kernel 2.6.23). The compiler used is g++4.1.2; all configurations being compiled with -DNDEBUG -O2 flags (release mode with compiler optimizations enabled). CGAL 3.3.1 [5] is used along with an *exact predicate inexact construction* kernel [59, 117]. Each experiment was repeated 30 times, and the average is taken.

The initialization costs of the filtering algorithm accounting for less than 0.13% of the corresponding total running time of the experiments, we consider them as negligible.

5.4.1   Clustering

In several applications such as image compression, quadrature, and cellular biology, to name a few, the goal is to partition a set of objects into k clusters, following an optimization criterion. Usually such criterion is the squared error function. Let S be a

variables	τ	α
$\{sa, sb, sc, sd, se\}$	$\leq 3L^2$	$\leq 8L^2\mu$
$\{a_1, b_1, c_1, a_2,$ $b_2, c_2, a_3, b_3, c_3\}$	$\leq 4L$	$\leq 4L\mu$
$\{d_1, d_2, d_3\}$	$\leq 6L^2$	$\leq 2\alpha_{\{sa,\ldots\}} + 2\tau_{\{sa,\ldots\}}\mu$ $\leq 16L^2\mu + 6L^2\mu \leq 22L^2\mu$
$\{m01, m02, m12, x01, x02, x12,$ $y01, y02, y12, z01, z02, z12\}$	$\leq 32L^2$	$\leq 4\alpha_{\{a_1,\ldots\}}\tau_{\{a_1,\ldots\}} + 4a_1{}^2\mu$ $\leq 64L^2 + 64L^2$ $\leq 128L^2\mu$
$\{g\}$	$\leq 384L^3$	$\leq 3\left(\alpha_{\{m01,\ldots\}}\tau_{c_3} + \alpha\{a_1,\ldots\}\tau_{\{m01,\ldots\}}\right)$ $+8\tau_{\{a_1,\ldots\}}\tau_{\{m01,\ldots\}}\mu$ $\leq 3(512L^3\mu + 128L^3\mu) + 1,024L^3\mu$ $\leq 2,944L^3\mu$
$\{x012, y012, z012\}$	$\leq 576L^4$	$\leq 3\left(\alpha_{\{m01,\ldots\}}\tau_{\{d_1,\ldots\}} + \alpha\{d_1,\ldots\}\tau_{\{m01,\ldots\}}\right)$ $+8\tau_{\{d_1,\ldots\}}\tau_{\{m01,\ldots\}}\mu$ $\leq 3(768L^4\mu + 576L^4\mu) + 1,536L^4\mu$ $\leq 5,568L^4\mu$

Table 5.2: **Certified 3D tolerance computation.** *Error bounds for the computation in 3D of the squared tolerance of a bi-cell.*

measurable set of objects in $\mathcal{Z}$ and $\mathcal{P} = \{S_i\}_1^k$ a k-partition of S. The squared error function associated with $\mathcal{P}$ is defined as:

$$\sum_{i=1}^{k}\int_{S_i} dist(p, \mu_i)^2\mathrm{d}p, \qquad (5.4)$$

where μ_i is the centroid of S_i and $dist$ is a given distance between two objects in $\mathcal{Z}$.

One relevant algorithm to find such partitions is the *k-means* algorithm. The most common form of the algorithm uses the Lloyd iterations [162, 189, 107, 176]. The Lloyd iterations algorithm starts by partitioning the input domain into k arbitrary initial sets. It then (i) calculates the centroid of each set and (ii) constructs a new partition by associating each point to the closest centroid; step (i) followed by step (ii) corresponds to one *Lloyd iteration*. The algorithm then successively applies Lloyd iterations until convergence.

When $S = \mathbb{R}^d$, $dist$ is the Euclidean distance, and ρ a density function[4] with a support in $\mathbb{R}^d$, the Lloyd iterations algorithm becomes as follows: First, compute the *Voronoi diagram* of an initial set of k random points following the distribution ρ. Next, each cell of the Voronoi diagram is integrated so as to compute its center of mass. Finally, each point is relocated to the centroid of its Voronoi cell. Note that, for each iteration, the Delaunay triangulation of the points is updated and hence each iteration is considered as a distinct time stamp.

Under the scheme described above, the convergence of Lloyd iterations is proven for $d = 1$ [107]. Although only weaker convergence results are known for $d > 1$ [189], the Lloyd iterations algorithm is commonly used for dimensions higher than 1 and experimentally converges to "good" point configurations.

---

[4]The unique purpose of the density function here is to compute centroids.

We consider in $\mathbb{R}^2$ one uniform density function ($\rho_1 = 1$) as well as three non-uniform density functions: $\rho_2 = x^2 + y^2$, $\rho_3 = x^2$ and $\rho_4 = \sin^2\sqrt{x^2 + y^2}$. We apply the Lloyd iterations to obtain evenly-distributed points in accordance with the above-mentioned density functions, see Figure 5.5.

The standard annuli widths and tolerances increase up to convergence, while the average displacement size quickly decreases, see Figure 5.6. In addition, the number of near-degenerate cases tends to decrease along with the iterations.

(a) Uniform density: $\rho = 1$

(b) Non-uniform density: $\rho = x^2 + y^2$

(c) Non-uniform density: $\rho = x^2$

(d) Non-uniform density: $\rho = \sin^2\left(\sqrt{x^2 + y^2}\right)$
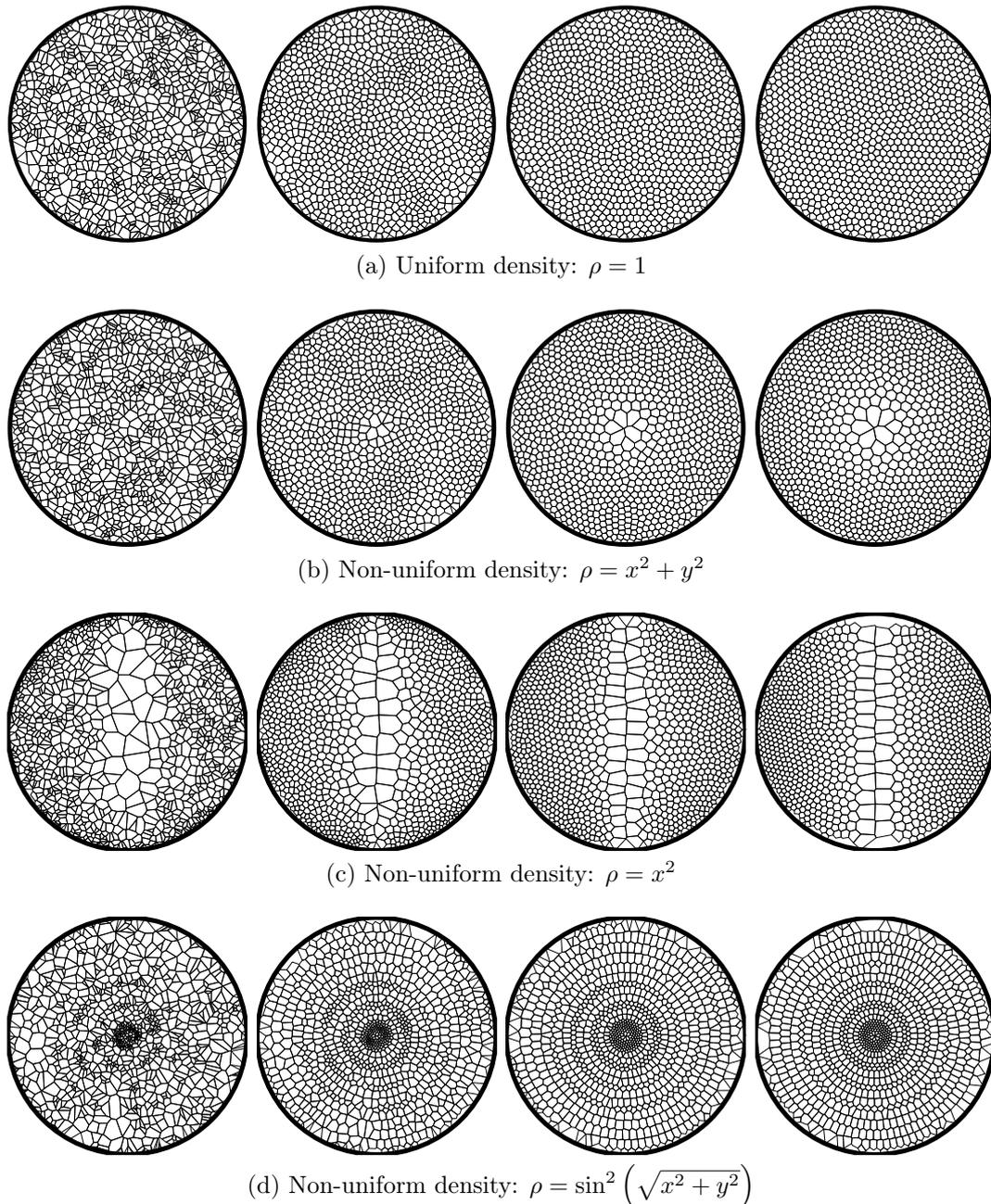
Figure 5.5: **Point distribution before and after Lloyd's iteration.** *1,000 points are sampled in a disc with (a) uniform density, (b) $\rho = x^2 + y^2$, (c) $\rho = x^2$, and (d) $\rho = \sin^2\sqrt{x^2 + y^2}$. The point sets are submitted to 1,000 Lloyd iterations with their respective density function. Pictures represent from left to right the 1st, 10th, 100th and 1000th iteration.*

As shown by Figure 5.7 and Table 5.3 the proposed filtering algorithm outperforms both the relocation and rebuilding algorithms. When we go further on the number of iterations, filtering becomes several times faster than rebuilding even though being a dynamic algorithm. It is worth saying that other strategies for accelerating the Lloyd iterations exist. We can cite, e.g., the Lloyd-Newton method, devised to reduce the overall number of iterations [105, 106, 161].
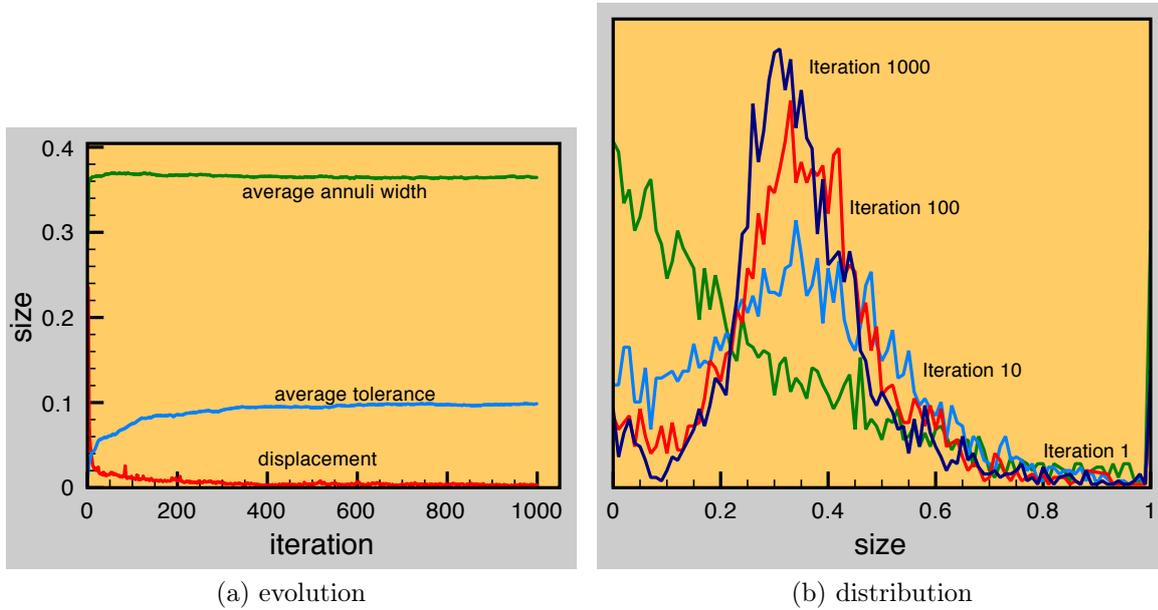
Figure 5.6: **Statistics in 2D.** *Consider a disc with a surface $= n$. The square root of this quantity is the unity of distance. The curves depict for $1,000$ Lloyd iterations with $\rho = x^2$: (a) the evolution of the average displacement size, the average standard annulus width and the average tolerance of vertices; (b) the distribution of the standard annulus width for iteration 1, 10, 100 and $1,000$.*

density	rebuilding		reloc.	filtering	
	(a)	(b)	(a/b)	(a)	(b)
uniform	2.63	2.63	1.00	16.05	40.8
$x^2 + y^2$	2.63	2.63	1.00	6.39	12.76
x^2	2.70	2.70	1.00	6.05	14.41
$\sin^2 \sqrt{x^2 + y^2}$	2.63	2.78	1.00	7.42	13.97

Table 5.3: **Speedup factors: K-means with Lloyd's iteration.** *The numbers listed represent the speedup factor of each algorithm with respect to the relocation algorithm, for a given input. Relocation and filtering are dynamic. (a) is the speed-up from the beginning to iteration $1,000$, (b) is the speed-up at last iteration.*

5.4.2   Mesh Optimization

To evaluate the filtering algorithm in 3D, we experiment it on an isotropic tetrahedron mesh generation scheme [217] based on a combination of Delaunay refinement and optimization. We focus on the optimization part of the algorithm, based upon an extension of

(a) uniform ($\rho = 1$)

(b) $\rho = x^2 + y^2$

(c) $\rho = x^2$
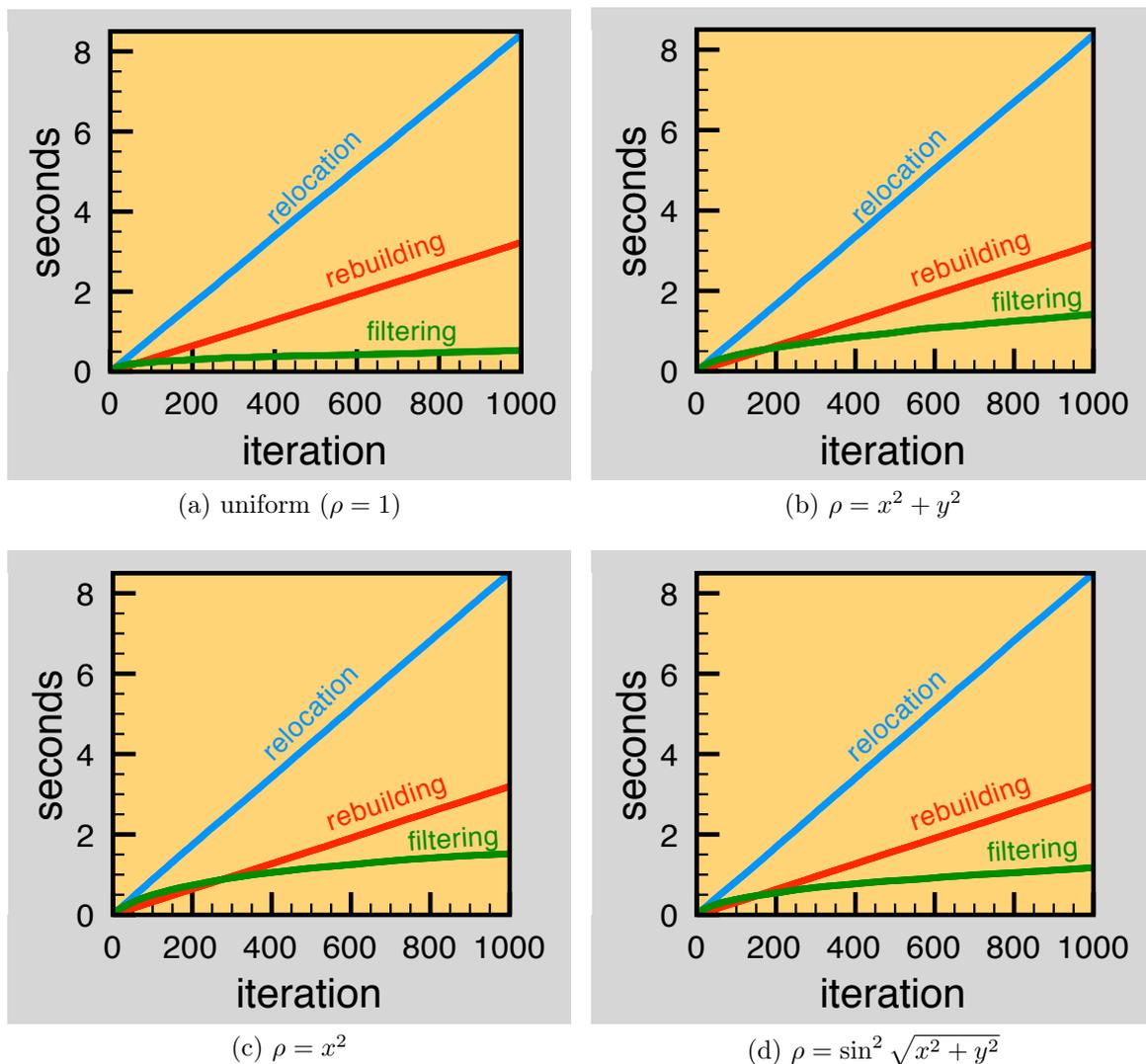
(d) $\rho = \sin^2 \sqrt{x^2 + y^2}$

Figure 5.7: **Computation times in 2D.** *The curves depict the cumulated computation times for running up to 1,000 Lloyd iterations with: (a) uniform density ($\rho = 1$); (b) $\rho = x^2 + y^2$; (c) $\rho = x^2$; and (d) $\rho = \sin^2 \sqrt{x^2 + y^2}$. The filtering algorithm consistently outperforms rebuilding for every density function. Note how the slope of the filtering algorithm's curve decreases over time.*

the Optimal Delaunay Triangulation approach [67], denoted by NODT for short. We first measure how the presented algorithm accelerates the optimization procedure, assuming that all vertices are relocated. We consider the following meshes:

- SPHERE: Unit sphere with $13,000$ vertices; see Figure 5.8(a).

- MAN: Human body with $8,000$ vertices; see Figure 5.8(b).

- BUNNY: Stanford Bunny with $13,000$ vertices; see Figure 5.8(c).

- HEART: Human heart with $10,000$ vertices; see Figure 5.8(d).

Figure 5.9 shows the MAN mesh model evolving over $1,000$ iterations of NODT optimization. (The mesh is chosen intentionally coarse and uniform for better visual de-

(a) SPHERE
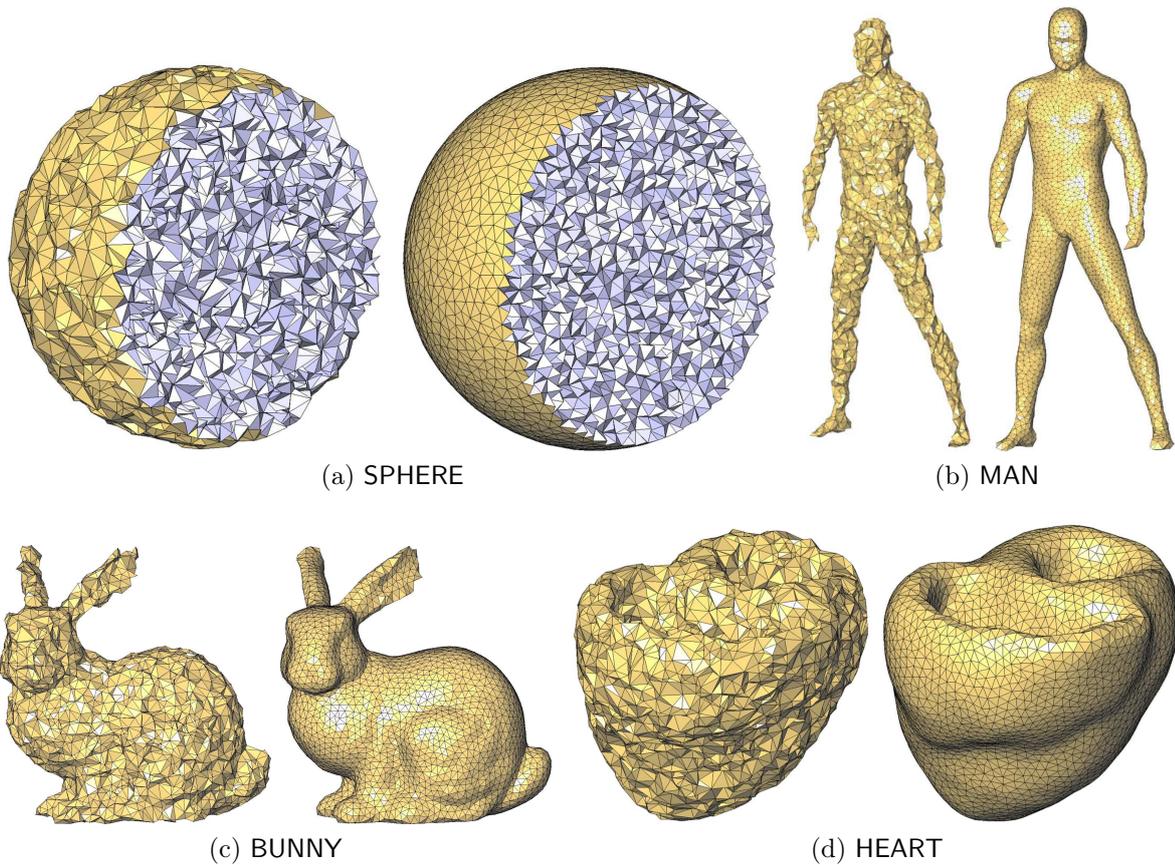
(b) MAN

(c) BUNNY

(d) HEART

Figure 5.8: **Mesh optimization based on Optimal Delaunay Triangulation.***(a) Cut-view on the SPHERE initially and after* $1,000$ *iterations; (b) MAN initially and after* $1,000$ *iterations; (c) BUNNY initially and after* $1,000$ *iterations; (d) HEART initially and after* $1,000$ *iterations.*

piction.) The figure depicts how going from 100 to $1,000$ iterations brings further improvements on the mesh quality. (Such improvements are confirmed over distribution of dihedral angles and over number of remaining slivers [217].) In meshes with variable sizing the improvement typically translates into 15% less leftover slivers. These experiments explain our will at accelerating every single optimization step.

Experiments in 3D show a good convergence of the average standard annulus width and average tolerance of vertices for the NODT mesh optimization process. However, the average tolerance of vertices converges in 3D to a proportionally smaller value than the average standard annulus width compared with the 2D case; see Figures 5.6(a) and 5.10(a). This is an effect of increasing the dimension and can be explained as follows: The average number of bi-cells including a given vertex is larger than 60 in three dimensions; this should be contrasted with the two-dimensional case, which is 12. As explained in Section 5.1.3, the tolerance of a vertex v is half the minimum of the standard annulus width of its adjacent bi-cells. In other words, the tolerance of a vertex is proportional to the minimum value of around 60 distinct standard annulus widths. Figure 5.10(b) quantizes how the standard deviation of the standard annulus widths in three dimensions is larger than in two dimensions. Figures 5.11(a) and 5.11(b) show respectively the percentage of failures and the amount of tolerance updates per failure in two and three dimensions.

<center>(a)   (b)   (c)</center>
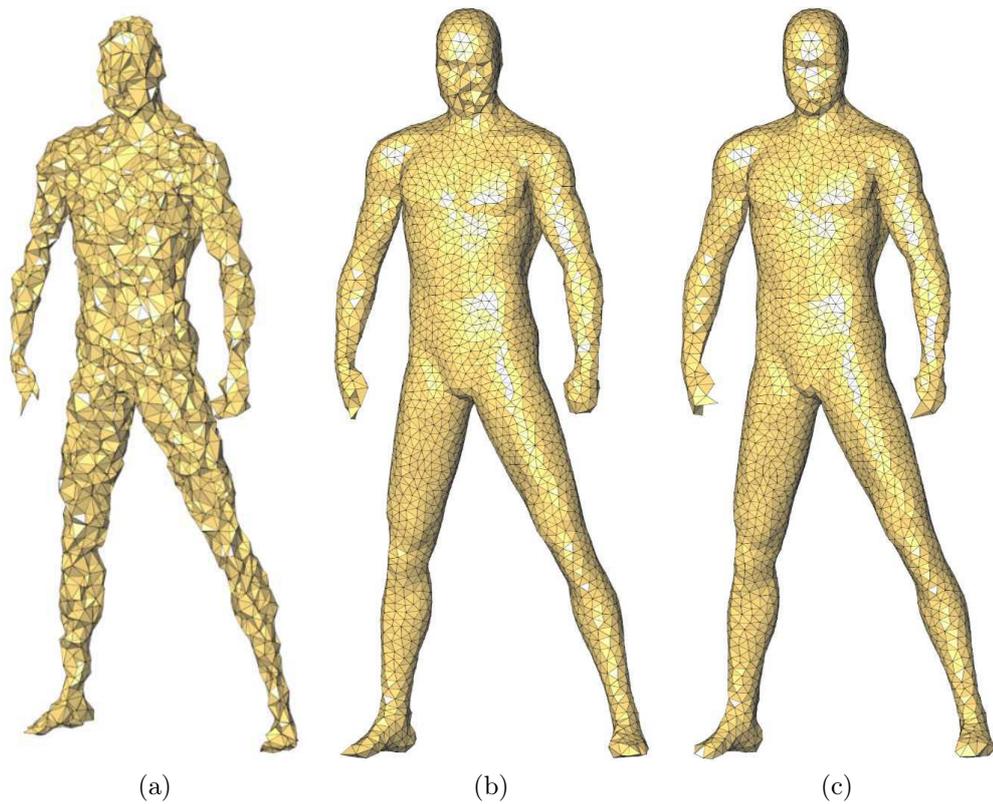
Figure 5.9: **Mesh quality improvement.** *(a) MAN initially; (b), (c) MAN after* 100 *and* 1,000 *iterations respectively.*

The rebuilding algorithm is considerably harder to outperform in three dimensions for two main reasons: (i) the deletion operation is dramatically slower, and (ii) the number of bi-cells containing a given point is five times larger than in two dimensions. Nevertheless, the filtering algorithm outperforms rebuilding for most input data considered in our experiments. In both two and three dimensions, it accelerates when going further on the number of iterations; see Figure 5.7, Figure 5.12, Table 5.3, and Table 5.4. In three dimensions, one recurring issue is the *persistent quasi-degenerate cases*. As the name suggests such cases consist of almost co-spherical configurations of the vertices of a bi-cell that persist during several iterations; this leads to several consecutive filter failures, which themselves trigger expensive point relocations. In our experiments, the amount of such persistent cases are negligible in 2D, but certainly not negligible in 3D; see Figure 5.11(c) and Figure 5.11(d).

Last but not least, we implemented a small variation of the tolerance algorithm suggested in Section 5.2.2. The latter consists of rebuilding for the first few iterations before switching to the filtering algorithm. The switching criterion is heuristic. Experimentally, we found that generally when 75% of the vertices remain inside their tolerance (55% in three dimensions), the performance of the filtering algorithm is more or less the same as rebuilding. This percentage can be used as a switching criterion. In our implementation we sample 40 random vertices at each group of four iterations and compute their tolerance. We compare these tolerances with their displacement sizes, and check if at least

	(a) evolution	(b) distribution
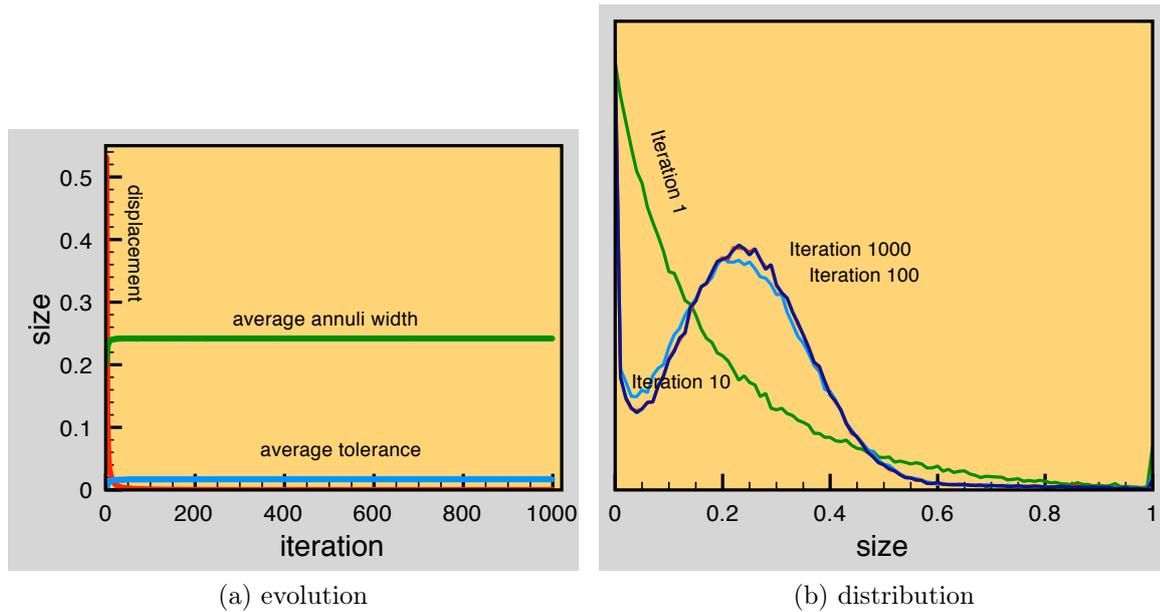
Figure 5.10: **Statistics in 3D.** *Consider a ball with volume* $= n$. *The cubic root of this quantity is the unity of distance. The figures depict* $1,000$ *iterations of the meshing optimization process on* SPHERE: *(a) evolution of the average displacement size, average standard annulus width and average tolerance of vertices over* $1,000$ *iterations; (b) distribution of the standard annulus width for* 1, 10, 100 *and* $1,000$ *iterations.*

density	rebuilding		reloc.	filtering	
	(a)	(b)	(a/b)	(a)	(b)
SPHERE	5.23	6.11	1.00	7.04	14.45
MAN	8.63	9.83	1.00	9.24	30.57
BUNNY	8.32	8.70	1.00	6.61	7.66
HEART	8.82	10.14	1.00	10.04	20.28

Table 5.4: **Speedup factors: NODT.** *The numbers listed represent the speedup factor of each algorithm with respect to the relocation algorithm, for a given input. (a) is the speed-up factor from the beginning to iteration* $1,000$, *(b) is the speed-up factor at the last iteration.*

75% of those vertices have their tolerance larger than their displacement size. In the positive case, we switch from the rebuilding algorithm to the filtering algorithm. With this variant of the filtering algorithm, which is not dynamic for the first few iterations, we obtain an improvement in the running time of 10% in the mesh optimization process of the HEART model.

Mesh optimization schemes such as NODT [217] are very labor-intensive due to the cost of computing the new point locations and of moving the vertices. However, as illustrated by Figure 5.9, a large number of iterations provides us with higher quality meshes. When optimization is combined with refinement, performing more iterations requires not only reducing computation time per iteration, but also additional experimental criteria. One example is the lock procedure which consists of relocating only a fraction of the mesh vertices [217]. In this approach, the locked vertices are the ones that are incident to only high quality tetrahedra (in terms of dihedral angles). Each time a vertex move or a
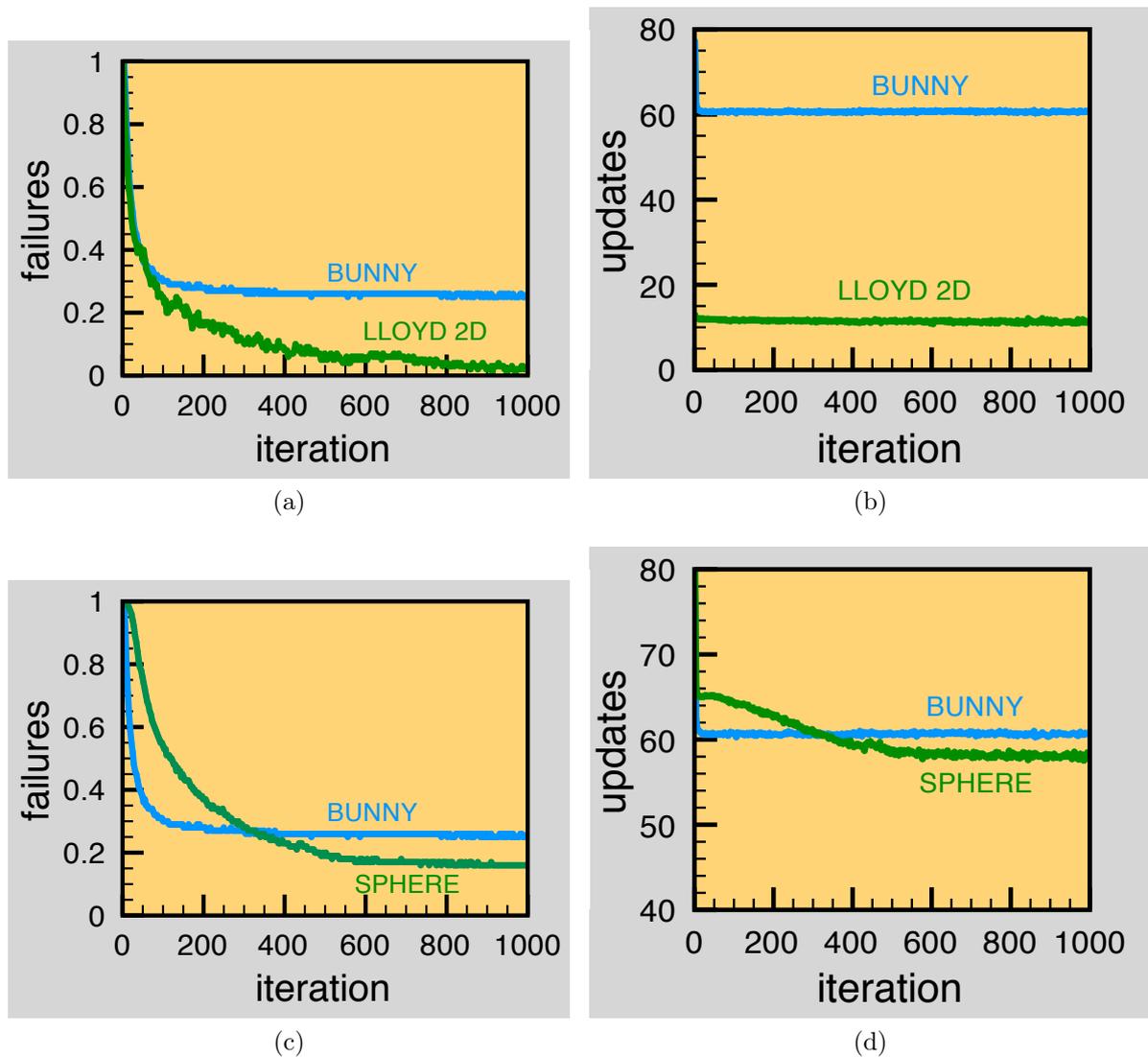
Figure 5.11: **Filter failures and number of tolerance updates.** *Consider the execution of the filtering algorithm for: the two-dimensional data set of Lloyd iterations with $\rho = x^2$; BUNNY and SPHERE. The curves depict the percentage of relocations for which the filter fails along the iterations, for (a) BUNNY and Lloyd's iteration with $\rho = x^2$, (c) BUNNY and SPHERE; the average number of tolerance updates done per filter failure, for (b) BUNNY and Lloyd's iteration with $\rho = x^2$, (d) BUNNY and SPHERE. Observe how the complexity is higher for the three-dimensional cases. Also note how the number of filter failures is higher for BUNNY compared to SPHERE (around 25% for BUNNY against 16% for SPHERE at the $1000th$ iteration).*

Steiner vertex is inserted into the mesh so as to satisfy user-defined criteria by Delaunay Refinement (sizing, boundary approximation error, element quality), all impacted vertices are unlocked. Our experiments show that more and more vertices get locked as the refinement and optimization procedures go along, until 95% of them are locked. In this context the dynamic approach is mandatory as the mesh refinement and optimization procedure may be applied very locally where the user-defined criteria are not yet satisfied. Note also that the status of each vertex evolves along the iterations as it can be unlocked
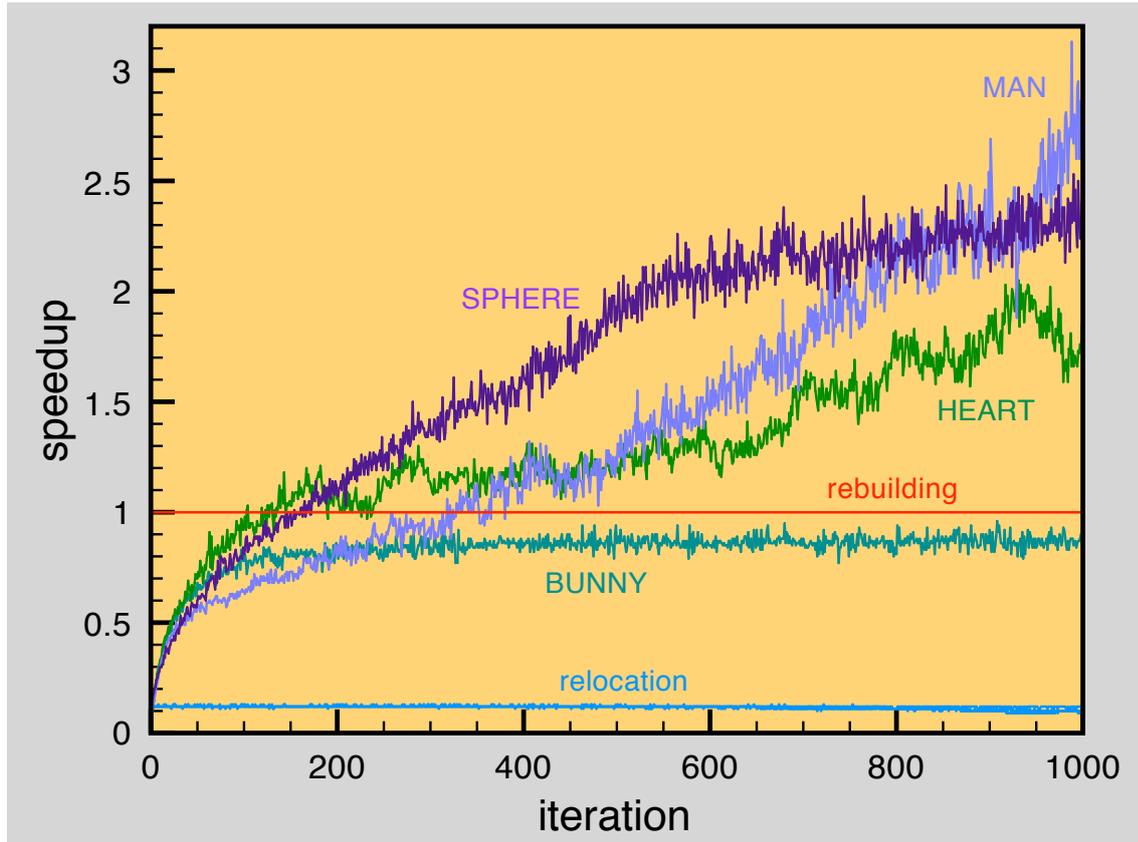
Figure 5.12: **Speedup factors: NODT.** *The figure represents the speedup factor of each algorithm with respect to the rebuilding algorithm along the iterations, for a given input. Names with capital letters (e.g., "SPHERE") in the figure, means the filtering algorithm working in the respective input data (e.g., SPHERE). The "speedup" of the relocation algorithm with respect to rebuilding is more or less constant for each input data.*

by the relocation of its neighbors. A static approach would slow down the convergence of the optimization scheme. In this context, accelerating the dynamic relocations makes it possible to go further on the number of iterations without slowing down the overall convergence process, so as to produce higher quality meshes.

5.5   Conclusion

We dealt with the problem of updating Delaunay triangulations for moving points in practice. We introduced the concepts of tolerance region and safe region of a vertex, and put them at work in a dynamic filtering algorithm that avoids unnecessary insert and remove operations when relocating the vertices.

We conducted several experiments to showcase the behavior of the algorithm for a variety of data sets. These experiments showed that the algorithm is particularly suitable for when the magnitude of the displacement keeps decreasing while the tolerances keep increasing. Such configurations occur in convergent schemes such as the Lloyd iterations. For the latter, and in two dimensions, the algorithm presented performs an order of magnitude faster than rebuilding during the last iterations.

In three dimensions, and although rebuilding the whole triangulation at each time stamp can be as faster as our algorithm when all vertices move, our solution is fully dynamic and outperforms previous dynamic solutions. Such a dynamic property is required for practical variational mesh generation and optimization techniques. This result makes it possible to go further on the number of iterations so as to produce higher quality meshes.

The parallelism being the pillar nowadays for huge speed-ups, it is important to note that the filtering portion of Algorithm 1 is easily parallelizable. More precisely, (i) checking whether a point is inside a tolerance region, or (ii) updating tolerance regions, are easily parallelizable tasks. Combinatorial changes however are not [42]. If points tend to destroy the Delaunay property at the same specific region, for general geometry, the best practical algorithms [46, 42] cannot achieve an speed-up factor proportional to the number of processors.

5.6   Open Problems

Problem 18. *Is there in three dimensions a dynamic filtering algorithm which performs an order of magnitude faster than rebuilding in practice?*

Regarding that question, we are rather pessimistic. For several triangulations, even checking statically their validity is not an order of magnitude faster than just rebuilding the triangulation. Actually, checking the validity of a triangulation dynamically: i.e for each vertex v of the triangulation, checking whether the star of v is valid; is slower than just rebuilding the whole Delaunay triangulation. A better dynamic filtering algorithm must surpasses ours on the following qualities: (i) tolerance regions should be bigger, (ii) tolerance regions should be faster to compute, (iii) it should be fast to check whether a point lies inside a tolerance region, and (iv) tolerance regions should be faster to update. We have tried without success some approaches:[5] (i) Algorithm 1 with the tolerance region replaced by the largest ball centered at v touching the boundary of the safe region at d points (instead of one point); (ii) Algorithm 2 with scaled and translated annuli to adapt to the speed of the displacements; (iii) Algorithm 2; (iv); Algorithm 1 and in case of failure Algorithm 2 (v) a variation of Algorithm 2 considering first order approximations. The later being "just" about twice worse than Algorithm 1.

A dynamic algorithm, such as Algorithm 1, has typically a **local** vision of what is happening in the input set; whereas a static algorithm has also a **global** vision. If a static algorithm is enough for a given application, than some interesting features become possible. First, usually in physical simulations, the triangulation of the points, up to isometries, barely changes between two time stamps, whereas the size of the displacements can be very large (typically during a translation of the input points). While a dynamic approach can't take that in consideration, a static one can. The same applies for scales (e.g., explosion simulation, gravitation simulation) and rotations. Also in a static setting, instead of looking at a vertex and its star, we can turn our attention to bi-cells and their tolerances. Such change in paradigm decreases the number of redundant updates that is required by the dynamic approach.

---

[5]Approaches are ordered from the worst to the "not so bad".

These new possibilities lead to the following questions:[6]

Problem 19. *Is there in three dimensions a static filtering algorithm that performs an order of magnitude faster than rebuilding in practice?*

Problem 20. *Is there a static filtering algorithm that is in some sense invariant to translations, scales and rotations?*

Finally, Algorithm 1 can be seen in a more abstract way, where (i) Delaunay triangulation corresponds to a data structure $\mathcal{A}$ (AVL trees, convex hulls, kd-trees, to name a few), (ii) points correspond to primitives, and (iii) tolerance region of a vertex in a Delaunay triangulation corresponds to the tolerance region of a primitive with respect to $\mathcal{A}$. In the same way kinetic data structures have been formalized to support a general kinetic framework (see Section 4.1.2), we believe that a more general *time stamp relocation data structure* supports a general time stamp relocation framework. The measure of quality of such time stamp relocation data structures is related to: (i) *memory*, the amount of additional space the filter requires; (ii) *region size*, the ratio between the tolerance region and the biggest region where primitives can move without invalidating the structure, i.e., the ratio between the false negatives and the total filter failures in average; (iii) *update time*, the time to update the structure when a filter failure occurs; (iv) *check time*, the time to check whether a point lies inside the tolerance region. We summarize this discussion with the following question:

Problem 21. *Is the Vertex Tolerance Filtering algorithm an instance of a more general time stamp point relocation framework?*

---

[6]A very recent work including techniques derived from this work's contribution, attempts to partially answer these questions [224].

Part III

Point location in triangulations

Chapter 6

On Some Greedy Spanning Trees embedded in $\mathbb{R}^d$

"Reductio ad absurdum, which Euclid loved so much, is one of a mathematician's finest weapons. It is a far finer gambit than any chess play: a chess player may offer the sacrifice of a pawn or even a piece, but a mathematician offers the game." — Godfrey Harold Hardy.

- P. M. M. de Castro and O. Devillers. On the Asymptotic Growth Rate of Some Spanning Trees Embedded in $\mathbb{R}^d$. Submitted (accepted), *Operation Research Letters*, 2010. (Also available as: Research Report 7179, INRIA, 2010.)

This chapter proposes to review some well-known trees embedded in $\mathbb{R}^d$, and present new results on the total weight of the Euclidean minimal k-insertion tree ($EMIT_k$) and some stars embedded in $\mathbb{R}^d$. Further than presenting new results on the growth rate of trees embedded in $\mathbb{R}^d$, this chapter builds some necessary fundamentals for Chapter 8.

A star is a tree having one vertex that is linked to all others, and $EMIT_k$ is based on minimal *sequential* insertions. More precisely, a point q is inserted in $EMIT_k$ by linking q to the point p that minimizes $f(\|p - q\|)$ amongst the k previously inserted points. Here, we are interested in functions $f(l) \sim l^\alpha$ as $l \to 0$. When $k \geq n - 1$, $EMIT_k$ is simply the minimal insertion tree $EMIT$ defined in [213]. For some well-chosen sequence of n points $EMIT$ coincides with the Euclidean minimum spanning tree ($EMST$) of these points. $EMIT$ is interesting because: (i) it can be constructed without the *a priori* knowledge of the whole sequence of points; (ii) the sum of its weight is sub-linear in n. This is not true for stars.

6.1 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) [207], which are related with the point location strategies
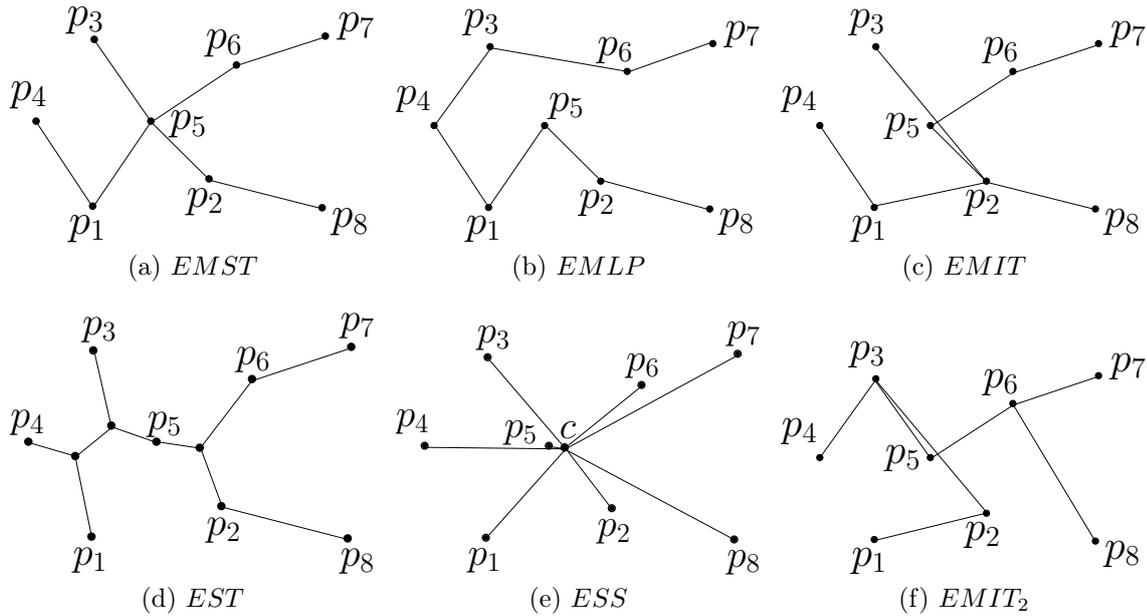
Figure 6.1: **Trees embedded in $\mathbb{R}^d$.**

developed in Chapter 8. Let $S = \{p_i, 1 \le i \le n\}$ 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 $\|p_i - p_j\|$. We denote the sum of the length of the edges of G by $\|G\|$. (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 6.1(a). $EMST$ can be computed with a greedy algorithm at a polynomial complexity.

EMLP. If instead of looking for a tree, we look for a path with minimal length spanning S, we get the *Euclidean minimum length path* denoted by $EMLP(S)$; see Figure 6.1(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

$$\|EMST(S)\| \le \|EMLP(S)\| < \|ETST(S)\| < 2\|EMST(S)\| \qquad (6.1)$$

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 $S_i = \{p_{\sigma(j)} \in S, i \le j\}$ with $T_1 = \{v_{\sigma(1)}\}$, $T_i = T_{i-1} \cup \{v_{\sigma(i)} v_{\sigma(j)}\}$ and a fixed k, with $1 \le k < n$, such that $v_{\sigma(i)} v_{\sigma(j)}$ has the shortest length for any $\max(1, i - k) \le j < i$. This tree is called the *Euclidean minimum k-insertion tree*, and is denoted by $EMIT_k(S)$; see Figure 6.1(f). When $k = n - 1$, we write $EMIT(S)$; see Figure 6.1(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 6.1(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 6.1(e).

6.1.1 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 [43] states that if p_i are independent and identically distributed random variables with compact support, then $\|ETST(S)\| = O(n^{1-1/d})$ with probability 1. By Eq.(6.1) the same bound is obtained for $\|EMLP(S)\|$ 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 proves [214] that the complexity of these graphs remains bounded by $O(n^{1-1/d})$ 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 $\sum_{i=2}^{n} \|p_{i-1}p_i\|$. Let $V_\sigma = \{p_{\sigma(1)}, p_{\sigma(2)}, \ldots, p_{\sigma(n)}\}$ be a sequence of n 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 [181] proved that in two dimensions the length of the path made by visiting S_σ sequentially is a $O(\log n)$ approximation of $\|ETST(S)\|$, and hence $\sum_{i=2}^{n} \|p_{\sigma(i-1)}p_{\sigma(i)}\| = O(\sqrt{n}\log n)$. One of the main interests of such heuristic is that σ can be found in $O(n \log n)$ time.

Finally, Steele shows that the growth rate of $\|EMIT(S)\|$ is as large as the one of $\|EMST(S)\|$ [213]; i.e., $\|EMIT(S)\| = O(n^{1-1/d})$.

6.2 A More General Framework: Weighted Distance

Consider now a more generic geometric graph $G = (V, E)$, where the edges $e_{ij} \in E$ linking two vertices v_i and v_j is weighted by $\|p_i - p_j\|^\alpha$, the *weighted distance* of p_i and p_j, i.e., its Euclidean length to the power of α. We denote the sum of the weights of the edges of G by $\|G\|_\alpha$ (remember that we simply denote $\|G\|$ for $\alpha = 1$). We also refer to $\|G\|_\alpha$ as the *weighted length* of G.

Concerning $\|EMST(S)\|_\alpha$, Steele proves [212] that if p_i are independent and identically distributed random variables with compact support, then $\|EMST(S)\|_\alpha = O(n^{1-\alpha/d})$ with probability 1. And, for the extreme case of $\alpha = d$, Aldous and Steele [21] show that $\|EMST(S)\|_d = O(1)$ if points are evenly distributed in the unit cube. Yukich [222] and Lee [157] determined the worst-case bounds of $\|EMST(S)\|_\alpha$ when $\alpha > 0$.

In the next section, we present some new results on $\|EMIT(S)\|$ and $\|EMIT_k(S)\|$ within the (more general) scenario described above.

6.3   New Results on Minimum Insertion Trees

In this section, we study the length of $EMIT(S)$ and $EMIT_k(S)$, when edges e_{ij} are weighted as an increasing function $f(l) \sim l^\alpha$ as $l \to 0$, with $l = \|e_{ij}\|$. Naturally, the combinatorics of $EMIT(S)$ or $EMIT_k(S)$ is invariant to such edge weights, since f is increasing, though the weighted length varies. The contributions in this section are three-fold. **First**, we extend the result of Steele [213] from $\alpha = 1$ to $\alpha \in (0, d)$, so as to obtain the following worst-case bound for any sequence of points in the unit cube: $\|EMIT(S)\|_\alpha = O(n^{1-\alpha/d})$ (Section 6.3.1); we also generalize this bound for $\|EMIT_k(S)\|_\alpha$. Results on the worst-case growth rate of $\|EMIT(S)\|_\alpha$ works regardless of the shape of the domain (here, the unit cube): Increasing the size of the cube to include sequence of points lying outside the unit cube, only affects the constants in the Big-O notation. **Second**, we obtain the expected total weight of $\|EMIT(S)\|_\alpha$ and $\|EMIT_k(S)\|_\alpha$, when $\alpha > 0$ and points are evenly distributed inside the unit ball (Section 6.3.2). **Third**, we compare the expected size of stars with fixed center, and also $\|EMIT_1(S)\|_\alpha$, when points are evenly distributed inside the unit ball (Section 6.3.3).

6.3.1   Worst-Case Growth Rate of $EMIT_k(S)$

We extend the result of Steele [213] for $\|EMIT(S)\|_\alpha$ as follows:

Theorem 22. *Let S be any sequence of n points in $[0, 1]^d$, then we have that*

$$\|EMST(S)\|_\alpha \le \|EMIT(S)\|_\alpha \le \gamma_{d,\alpha} n^{1-\alpha/d},$$

with $d \ge 2$ and $0 < \alpha < d$. Where,

$$\gamma_{d,\alpha} = 1 + \frac{2^{4d} d^{d/2}}{(2^\alpha - 1)(d/\alpha - 1)}.$$

The proof of Theorem 22 follows the same line as Steele [213], and starts with the two lemmas below. Given a fixed sequence $S = \{p_1, p_2, \dots, p_n\}$ of points in $\mathbb{R}^d$, then we can build a spanning tree for S by sequentially joining p_i to the tree formed by $\{p_1, p_2, \dots, p_{i-1}\}$ for $1 < i \le n$. We join p_i to the tree, by adding the edge e linking p_i to $p \in \{p_1, p_2, \dots, p_{i-1}\}$, such that p minimizes $\|p_i - p_j\|^\alpha$ for $1 \le j < i$. Let $w_i \in \mathbb{R}$ be defined as follow:

$$w_i = \min_{1 \le j < i} \|p_i - p_j\|^\alpha, \tag{6.2}$$

then w_i is the minimal cost of joining p_i to a vertex of a spanning tree of $\{p_1, p_2, \dots, p_{i-1}\}$. Now, we have that $\|EMIT(S)\|_\alpha = \sum_{1 < i \le n} w_i$.

Lemma 23. *If $\{p_1, p_2, \dots, p_n\} \subset [0, 1]^d$ and $w_i = \min_{1 \le j < i} \|p_i - p_j\|^\alpha$, for $1 < i \le n$, $d \ge 2$ and $0 < \alpha < d$, then for any $0 < \lambda < \infty$ we have*

$$\sum_{\lambda \le w_i < 2^\alpha \lambda} w_i^{d/\alpha} \le 8^d d^{d/2}. \tag{6.3}$$

Proof. Let $C = \{i : \lambda \le w_i < 2^\alpha \lambda\}$ and for each $i \in C$ let B_i be a ball of radius $r_i = \frac{1}{4} w_i^{1/\alpha}$ with center p_i. We argue by contradiction that $B_i \cap B_j = \emptyset$ for all $i < j$.

If $B_i \cap B_j \neq \emptyset$, then the bounds $r_i \leq 2^\alpha \lambda$ and $r_j \leq 2^\alpha \lambda$ gives us

$$\|p_i - p_j\| \leq \frac{1}{4}(w_i^{1/\alpha} + w_j^{1/\alpha}) < \lambda^{1/\alpha}. \tag{6.4}$$

But, by definition of w_j we have $\|p_i - p_j\|^\alpha \geq w_j$ for all $i < j$, which implies $\|p_i - p_j\| \geq w_j^{1/\alpha}$ for all $i < j$; and, by the lower bound on the summands in Eq.(6.3) we have $\lambda \leq w_j$, which means $\lambda^{1/\alpha} \leq w_j^{1/\alpha}$, so we also see $\|p_i - p_j\| \geq \lambda^{1/\alpha}$. Since $\|p_i - p_j\| \geq \lambda^{1/\alpha}$ contradicts Eq.(6.4), we have $B_i \cap B_j = \emptyset$.

Now, since all of the balls B_i are disjoint and contained in a sphere with radius $2\sqrt{d}$, the sum of their volumes is bounded by the volume of the sphere of radius $2\sqrt{d}$. Thus, if ω_d denotes the volume of the unit ball in $\mathbb{R}^d$, we have the bound $\sum_{i \in C} \omega_d w_i^{d/\alpha} 4^{-d} \leq \omega_d 2^d d^{d/2}$ from which Eq.(6.3) follows. □

Lemma 24. *Let Ψ be a positive and non-increasing function on the interval $\left(0, \sqrt{d}\right]$, then for any $0 < a < b \leq \sqrt{d}$, with $d \geq 2$ and $0 < \alpha < d$,*

$$\sum_{a \leq w_i \leq b} w_i^{(d/\alpha)+1} \Psi(w_i) \leq \frac{2^\alpha}{2^\alpha - 1} \cdot 8^d d^{d/2} \int_{a/2^\alpha}^b \Psi(\lambda) d\lambda. \tag{6.5}$$

Proof. By Lemma 23 we have for any $0 < \lambda < \infty$,

$$\sum_{a \leq w_i < b} w_i^{d/\alpha} I(\lambda \leq w_i < 2^\alpha \lambda) \leq 8^d d^{d/2},$$

where

$$I(\lambda \leq w_i < 2^\alpha \lambda) = I(\frac{1}{2^\alpha} w_i \leq \lambda < w_i)$$

is the indicator function. If we multiply by $\Psi(\lambda)$ and integrate over $[\frac{1}{2^\alpha}a, b]$, we find

$$\sum_{a \leq w_i \leq b} w_i^{(d/\alpha)} \int_{w_i/2^\alpha}^{w_i} \Psi(\lambda) d\lambda \leq 8^d d^{d/2} \int_{a/2^\alpha}^b \Psi(\lambda) d\lambda. \tag{6.6}$$

Since Ψ is non-increasing, the integrand on the left-hand side of Eq.(6.6) is bounded from below by $\Psi(w_i)$, so $\Psi(w_i) w_i (1 - \frac{1}{2^\alpha}) \leq \int_{w_i/2^\alpha}^{w_i} \Psi(\lambda) d\lambda$, and Eq.(6.5) follows from Eq(6.6). □

Proof of Theorem 22. Divide the set $\{w_2, w_3, \ldots, w_n\}$ in two sets $R_1 = \{w_i : w_i \leq n^{-\alpha/d}\}$ and $R_2 = \{w_i : w_i > n^{-\alpha/d}\}$. We have the trivial bound $\sum_{w_i \in R_1} w_i = \sum_{w_i \leq n^{-\alpha/d}} w_i \leq n \cdot n^{-\alpha/d} = n^{1-\alpha/d}$. Now, let $[a, b] = \left[n^{-\alpha/d}, d^{\alpha/2}\right]$, $\Psi(\lambda) = \lambda^{-d/\alpha}$ in Eq.(6.5), then we have:

$$\sum_{w_i \geq n^{-\alpha/d}} w_i \leq \frac{2^\alpha}{2^\alpha - 1} \cdot 8^d d^{d/2} (d/\alpha - 1)^{-1} \cdot \left(2^{d-\alpha} n^{1-\alpha/d} - d^{1-\alpha/2}\right).$$

And hence, $\sum_{w_i \in R_2} w_i \leq \frac{2^{4d} d^{d/2}}{(2^\alpha - 1)(d/\alpha - 1)} \cdot n^{1-\alpha/d}$. Now, we have that

$$\sum_{i=2}^n w_i = \sum_{w_i \in R_1} w_i + \sum_{w_i \in R_2} w_i \leq \left(1 + \frac{2^{4d} d^{d/2}}{(2^\alpha - 1)(d/\alpha - 1)}\right) \cdot n^{1-\alpha/d},$$

from which Theorem 22 follows. The constant is $\gamma_{d,\alpha} = 1 + \frac{2^{4d}d^{d/2}}{(2^\alpha - 1)(d/\alpha - 1)}$.   □

Naturally, Theorem 22 generalizes for $EMIT_k$ for $0 < \alpha < d$ as follows:

Corollary 25. *Let S be any sequence of n points in $[0,1]^d$, then we have that $\|EMIT_k(S)\|_\alpha = O(n \cdot k^{-\alpha/d})$, with $d \geq 2$ and $0 < \alpha < d$.*

6.3.2   Expected Growth Rate of $EMIT_k(S)$

Now, we compute a bound on the expected weight of an edge of $EMIT_k(S)$ for points evenly distributed in the unit ball $\mathcal{B}$.

Theorem 26. *When points are independent and identically distributed random variables following the uniform distribution inside $\mathcal{B}$, the expected weight $E(weight)$ of an edge of $EMIT_k(S)$, with positive α, verifies:*

$$(\alpha/d)B(k+1,\alpha/d) \leq E(weight) \leq 2^\alpha(\alpha/d)B(k+1,\alpha/d), \qquad (6.7)$$

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

To prove Theorem 26, we evaluate the weighted distance between the origin and the closest amongst k points $\{p_1, p_2, \ldots, p_k\}$ evenly distributed in $\mathcal{B}$. This provides the lower bound. Then, we find an upper bound on the weighted distance between any point inside the ball and the closest amongst k points $\{p_1, p_2, \ldots, p_k\}$ evenly distributed in $\mathcal{B}$.

Lemma 27. *Let c be a point inside $\mathcal{B}$, $Prob\,(\|p - c\| \leq l) = P_c(l)$ be the probability that the distance between a point $p \in \mathcal{B}$ and c is less or equal to l, and $P_{c,k}(l) = Prob\,(\min_{1 \leq j \leq k}(\|p_j - c\|) \leq l)$ be the cumulative distribution function of the minimum distance among k points independent and identically distributed inside $\mathcal{B}$ and c, then*

$$P_{c,k}(l) = 1 - (1 - P_c(l))^k$$

Proof.

$$
\begin{aligned}
P_{c,k}(l) &= Prob\left(\min_{1 \leq j \leq k}(\|p_j - c\|) \leq l\right) = 1 - Prob\,(\|p_j - c\| > l\,, 1 \leq j \leq k)\\
&= 1 - Prob\,(\|p_1 - c\| > l)^k = 1 - (1 - P_c(l))^k. \quad □
\end{aligned}
$$

A direct consequence of Lemma 27 is the following corollary.

Corollary 28. *Let $P_{\mathcal{B},k}(l) = Prob\,(\min_{1 \leq j \leq k}(\|p_j - O\|) \leq l)$ be the cumulative distribution function of the minimum distance among k points independent and identically distributed following the uniform distribution inside $\mathcal{B}$, and the center of $\mathcal{B}$, then $P_{\mathcal{B},k}(l) = 1 - (1 - l^d)^k$.*

Lemma 29. *The expected value $E\left(\min_{1 \leq j \leq k} \|p_j - O\|^\alpha\right)$ of the minimum weighted distance among k points independent and identically distributed following the uniform distribution inside $\mathcal{B}$ and the center of $\mathcal{B}$, with positive α, is given by: $E\left(\min_{1 \leq j \leq k} \|p_j - O\|^\alpha\right) = (\alpha/d)B(k+1,\alpha/d)$.*

Proof. Using Corollary 28, we have:

$$
\begin{aligned}
E\left(\min_{1\leq j\leq k}\|p_j - O\|^\alpha\right) &= \int_0^1 l^\alpha P'_{\mathcal{B},k}(l)\mathrm{d}l = kd\int_0^1 l^{d-1+\alpha}\left(1-l^d\right)^{k-1}\mathrm{d}l \\
&= kB(k, 1+\alpha/d) = (\alpha/d)B(k+1, \alpha/d). \quad \square
\end{aligned}
$$

Now, we shall obtain the upper bound. First we obtain a general expression for the expected value of $\min_{1\leq j\leq k}\|p_j - c\|^\alpha$. Assume $\delta(c) = 1 + \|c - O\|$ in what follows.

Lemma 30. *The expected value $E\left(\min_{1\leq j\leq k}\|p_j - c\|^\alpha\right)$ of the minimum weighted distance among k points independent and identically distributed inside $\mathcal{B}$ and c, with positive α, is given by: $E\left(\min_{1\leq j\leq k}\|p_j - c\|^\alpha\right) = \int_0^{\delta(c)} \alpha l^{\alpha-1}(1 - P_c(l))^k\mathrm{d}l$.*

Proof. As $P_c(0) = 0$ and $P_c(\delta(c)) = 1$, integration by parts gives us the following identity:

$$
i \cdot \int_0^{\delta(c)} l^\alpha P'_c(l)P_c^{i-1}(l)\mathrm{d}l = \delta(c)^\alpha - \int_0^{\delta(c)} \alpha l^{\alpha-1}P_c^i(l)\mathrm{d}l, i > 0. \tag{6.8}
$$

From Lemma 27, we also have the following expression for $E\left(\min_{1\leq j\leq k}\|p_j - O\|^\alpha\right)$:

$$
\begin{aligned}
E\left(\min_{1\leq j\leq k}\|p_j - c\|^\alpha\right) &= \int_0^{\delta(c)} kl^\alpha P'_c(l)(1 - P_c(l))^{k-1}\mathrm{d}l \\
&= \sum_{i=0}^{k-1}(-1)^i\binom{k-1}{i}\int_0^{\delta(c)} kl^\alpha P'_c(l)P_c^i(l)\mathrm{d}l. \tag{6.9}
\end{aligned}
$$

Replacing Eq.(6.8) in Eq.(6.9) leads to:

$$
\begin{aligned}
\sum_{i=0}^{k-1}(-1)^i\binom{k-1}{i}\int_0^{\delta(c)} kl^\alpha P'_c(l)P_c^i(l)\mathrm{d}l &= \sum_{i=0}^{k}(-1)^i\binom{k}{i}\int_0^{\delta(c)} \alpha l^{\alpha-1}P_c^i(l)\mathrm{d}l \\
&= \int_0^{\delta(c)} \alpha l^{\alpha-1}(1 - P_c(l))^k\mathrm{d}l. \quad \square
\end{aligned}
$$

Proof of Theorem 26. Lemma 29 gives us the lower bound in Theorem 26. Now, for a point $c \in \mathcal{B}$, if we take a function $\Psi(l)$ such that $\Psi(l) \leq P_c(l)$ for $0 \leq l \leq \delta(c)$, it upper bounds the integral in Lemma 30. Take $\Psi(l) = (l/\delta(c))^d$, then we have:

$$
\begin{aligned}
E\left(\min_{1\leq j\leq k}\|p_j - c\|^\alpha\right) &= \int_0^{\delta(c)} \alpha l^{\alpha-1}(1 - P_c(l))^k\mathrm{d}l \\
&\leq \int_0^{\delta(c)} \alpha l^{\alpha-1}(1 - \Psi(l))^k\mathrm{d}l \\
&= \int_0^{\delta(c)} \alpha l^{\alpha-1}(1 - l^d/\delta(c)^d)^k\mathrm{d}l \\
&= \delta(c)^\alpha(\alpha/d)B(k+1, \alpha/d).
\end{aligned}
$$

And thus, $\delta(c) = 2$ (c on the boundary of $\mathcal{B}$) maximizes the value above. This completes the proof. $\square$

From Theorem 26, we have that the expected weight w_i of the i-th edge of $\|EMIT(S)\|_\alpha$ for points evenly distributed inside $\mathcal{B}$, is such that

$$(\alpha/d)B(i+1,\alpha/d) \le w_i \le 2^\alpha(\alpha/d)B(i+1,\alpha/d).$$

Evaluating for $n-1$ gives

$$(\alpha/d)\sum_{i=2}^{n} B(i,\alpha/d) \le \|EMIT(S)\|_\alpha \le 2^\alpha(\alpha/d)\sum_{i=2}^{n} B(i,\alpha/d).$$

By using Stirling's identity $B(x,y) \sim \Gamma(y)x^{-y}$ above, the expected growth rate of $\|EMIT(S)\|_\alpha$, with points in S evenly distributed in $\mathcal{B}$, is:

- $\Theta\left(n^{1-\alpha/d}\right)$, for $0 < \alpha < d$. It is noteworthy that there exists $n_0 < \infty$, such that for $n > n_0$, $\|EMIT(S)\|_\alpha$ is bounded by $(1+\epsilon)\Gamma(1+\alpha/d)2^\alpha n^{1-\alpha/d}$ with ϵ as small as we want. The constant is considerably smaller than in Theorem 22.
- $\Theta(\log n)$ for $\alpha = d$. One might contrast this growth rate with the growth rate of $\|EMST(S)\|_d$ in the cube, which is $O(1)$ [21].
- And $O(1)$ for $\alpha > d$.

Analogously, the expected growth rate of $\|EMIT_k(S)\|_\alpha$ for points evenly distributed inside $\mathcal{B}$ is:

- $\Theta\left(n \cdot k^{-\alpha/d}\right)$ for $0 < \alpha < d$.
- $\Theta\left(n \cdot \log k/k\right)$ for $\alpha = d$.
- And $\Theta(n/k)$ for $\alpha > d$.

6.3.3   Two Stars and a Path

Consider the unit ball $\mathcal{B}$, in this section, we consider the star centered at a given point $c \in \mathcal{B}$ and whose leaves are a set $S = \{p_i, 1 \le i \le n\}$ of n evenly distributed points in $\mathcal{B}$.[1] By addition of the expectation, the expected weighted length of a random edge of the star is the expected weighted length between c and a random point in $\mathcal{B}$

$$E(\|p - c\|^\alpha, p \in \mathcal{B}) = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{\|p_i - c\|^\alpha}{n}.$$

Denote the stars with shortest and largest expected weighted length inside the ball by $\mathcal{S}$ and $\mathcal{H}$ as $n \to \infty$ respectively. Let $E(\|p - O\|^\alpha, p \in \mathcal{B})$ and $E(\|p - \Omega\|^\alpha; p \in \mathcal{B})$ be the expected value of an edge of $\mathcal{S}$ and $\mathcal{H}$ respectively, then as $n \to \infty$ the size of $\mathcal{S}$ and $\mathcal{H}$ becomes arbitrarily close to $n \cdot E(\|p-O\|^\alpha, p \in \mathcal{B})$ and $n \cdot E(\|p-\Omega\|^\alpha; p \in \mathcal{B})$ respectively.

We analyze in the sequel the values of $E(\|p - O\|^\alpha, p \in \mathcal{B})$ and $E(\|p - \Omega\|^\alpha; p \in \mathcal{B})$.

Theorem 31. *When points are independent and identically distributed random variables following the uniform distribution inside $\mathcal{B}$, the expected weight $E(\|p - O\|^\alpha; p \in \mathcal{B})$ of an edge of the star centered at the center of the unit ball, with positive α, is given by:*

$$\left(\frac{d}{d+\alpha}\right).$$

---

[1]Stars in this section have a fixed center regardless of n.

Proof. Let $\mathcal{B}_l$ be a ball with radius l centered at the origin, we have

$$
\begin{aligned}
E(\|p - O\|^\alpha; p \in \mathcal{B}) &= \int_0^1 l^\alpha Prob(p \in \mathcal{B}_{l+dl} \setminus \mathcal{B}_l)dl \tag{6.10} \\
&= \int_0^1 dl^{d-1+\alpha}dl = \frac{d}{d+\alpha}. \quad \square
\end{aligned}
$$

Theorem 32. *When points are independent and identically distributed random variables following the uniform distribution inside $\mathcal{B}$, the expected weight $E(\|p - \Omega\|^\alpha; p \in \mathcal{B})$ of an edge of the star centered at the boundary of the unit ball, with positive α, is given by:*

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

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

For the computation of $E(\|p - \Omega\|^\alpha; p \in \mathcal{B})$ we need the following lemma.

Lemma 33. *Let Ω be a point on the boundary of the unit ball $\mathcal{B}_{unit}$, and $P_{\mathcal{H}}(l) = Prob(\|p - \Omega\| \leq l; p \in \mathcal{B}_{unit})$ be the cumulative distribution function of distances between an uniformly distributed random point inside $\mathcal{B}_{unit}$ and Ω, then*

$$
P_{\mathcal{H}}(l) = \frac{1}{B\left(\frac{d}{2} + \frac{1}{2}, \frac{1}{2} \right)} \left(\int_0^{\arccos(1 - l^2/2)} \sin^d(\lambda)d\lambda + l^d \int_0^{\arccos(l/2)} \sin^d(\lambda)d\lambda \right),
$$

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

Proof. If we denote $\mathcal{B}_l$ the ball of radius l centered in Ω, the desired probability is clearly $volume(\mathcal{B}_l \cap \mathcal{B}_{unit})/volume(\mathcal{B}_{unit})$. $\mathcal{B}_l \cap \mathcal{B}_{unit}$ is the union of two spherical caps limited by the hyperplane $x = 1 - l^2/2$. The volume of the spherical cap formed by crossing a ball $\mathcal{B}_R$ with radius R centered at the origin, with the hyperplane $x = R - h$, for $0 \leq h \leq 2R$, is given by $R^d \frac{\pi^{\frac{d-1}{2}}}{\Gamma(\frac{d+1}{2})} \int_0^{\arccos\left(\frac{R-h}{R}\right)} \sin^d(\lambda)d\lambda$. Computing the two volumes and summing them complete the proof. $\square$

Proof of Theorem 32. The theorem follows from:

$$
\begin{aligned}
E(\|p - \Omega\|^\alpha; p \in \mathcal{B}) &= \int_0^2 l^\alpha P_{\mathcal{H}}'(l)dl \\
&= \frac{1}{2} \int_0^2 \frac{l^{d+\alpha} \left(1 - \frac{l^2}{4} \right)^{\frac{d-1}{2}} dl}{B\left(\frac{d}{2} + \frac{1}{2}, \frac{1}{2} \right)} + \frac{1}{2} \int_0^2 2dl^{d-1+\alpha} \frac{\int_0^{\arccos(l/2)} \sin^d(\lambda)d\lambda}{B\left(\frac{d}{2} + \frac{1}{2}, \frac{1}{2} \right)} dl
\end{aligned}
$$

The right part of the expression above corresponds exactly to the expected value of l^α where l is the length of a random segment determined by two evenly distributed points in the unit ball [190]. Its value is given by:

$$
2^{d+\alpha} \left(\frac{d}{d+\alpha} \right) \frac{B\left(\frac{d}{2} + \frac{1}{2}, \frac{d}{2} + \frac{1}{2} + \frac{\alpha}{2} \right)}{B\left(\frac{d}{2} + \frac{1}{2}, \frac{1}{2} \right)}. \tag{6.11}
$$

The left part is simplified to obtain the following expression:

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

Finally, we have $E(\|p - \Omega\|^\alpha; p \in \mathcal{B}) = 2^{d+\alpha} \left(\frac{2d+\alpha}{2d+2\alpha}\right) \frac{B\left(\frac{d}{2}+\frac{1}{2}, \frac{d}{2}+\frac{1}{2}+\frac{\alpha}{2}\right)}{B\left(\frac{d}{2}+\frac{1}{2}, \frac{1}{2}\right)}.$ □

We may ask now what is the value of the ratio $\rho(d, \alpha)$ between $E(\|p - \Omega\|^\alpha; p \in \mathcal{B})$ and $E(\|p - O\|^\alpha; p \in \mathcal{B})$. It is an easy exercise to verify that $\rho(1, \alpha) = 2^\alpha$. In Corollary 34, we compute $\lim_{d \to \infty} \rho(d, \alpha)$.

Corollary 34. *The ratio* $\rho(d, \alpha) = \frac{E(\|p-\Omega\|^\alpha; p \in \mathcal{B})}{E(\|p-O\|^\alpha; p \in \mathcal{B})}$ *when* $d \to \infty$ *is given by* $2^{\alpha/2}$.

Proof. Computing $\rho(d, \alpha)$ with Theorems 31 and 32 gives:

$$\rho(d, \alpha) = 2^{d+\alpha}\left(\frac{2d+\alpha}{2d}\right) \frac{B\left(\frac{d}{2}+\frac{1}{2}, \frac{d}{2}+\frac{1}{2}+\frac{\alpha}{2}\right)}{B\left(\frac{d}{2}+\frac{1}{2}, \frac{1}{2}\right)} \tag{6.12}$$

Using Stirling's identities: $B(a, b) \sim \sqrt{2\pi} a^{a-\frac{1}{2}} b^{b-\frac{1}{2}}/(a+b)^{a+b-\frac{1}{2}}, a, b \gg 0$ and $B(a, b) \sim \Gamma(b) a^{-b}, a \gg b > 0$, we have:

$$
\begin{aligned}
\lim_{d \to \infty} \rho(d, \alpha) &= \lim_{d \to \infty} \left\{ 2^{d+\alpha}\left(\frac{2d+\alpha}{2d}\right) \frac{B\left(\frac{d}{2}+\frac{1}{2}+\frac{\alpha}{2}, \frac{d}{2}+\frac{1}{2}\right)}{B\left(\frac{d+1}{2}, \frac{1}{2}\right)} \right\} \\
&= \lim_{d \to \infty} \left\{ \frac{2^{d+\alpha}\sqrt{2\pi}\left(\frac{d}{2}+\frac{1}{2}\right)^{\frac{d}{2}}\left(\frac{d}{2}+\frac{1}{2}+\frac{\alpha}{2}\right)^{\frac{d}{2}+\frac{\alpha}{2}}}{\sqrt{\pi}\left(d+1+\frac{\alpha}{2}\right)^{d+\frac{1}{2}+\frac{\alpha}{2}}\left(\frac{d}{2}+\frac{1}{2}\right)^{-\frac{1}{2}}} \right\} \\
&= 2^{\alpha/2} \cdot \lim_{d \to \infty} \left\{ \frac{(d+1)^{\frac{d+1}{2}}(d+1+\alpha)^{\frac{d+\alpha}{2}}}{(d+1+\frac{\alpha}{2})^{\frac{d+1}{2}}(d+1+\frac{\alpha}{2})^{\frac{d+\alpha}{2}}} \right\} \\
&= 2^{\alpha/2} \cdot e^{-\alpha/4} \cdot e^{\alpha/4} = 2^{\alpha/2} \quad \square
\end{aligned}
$$

Let $E(\|p - p'\|^\alpha; p, p' \in \mathcal{B})$ be the expected weight of an edge of $EMIT_1$ (a random path in the unit ball). Then, $E(\|p - p'\|^\alpha; p, p' \in \mathcal{B})$ is the expected value of l^α, where l is the length of a random segment determined by two evenly distributed points in the unit ball. This is given by Eq.(6.11). From Theorem 31, Theorem 32, Corollary 34 and Eq.(6.11), we obtain the following corollary:

Corollary 35. *The ratios:*
- $E\left(\|p - p'\|^\alpha; p, p' \in \mathcal{B}\right)/E\left(\|p - O\|^\alpha; p \in \mathcal{B}\right)$ *is* $2^\alpha/(1+\alpha/2)$ *when* $d = 1$, *and* $2^{\alpha/2}$ *when* $d \to \infty$;
- $E\left(\|p - \Omega\|^\alpha; p \in \mathcal{B}\right)/E\left(\|p - p'\|^\alpha; p, p' \in \mathcal{B}\right) = 1 + \alpha/2d.$

6.4 Conclusion

In this chapter, we extended the result in Steele [213], providing new results in a more general framework. We also gave evidence that the worst-case constant in Theorem 22

$\left(\gamma_{d,\alpha} = 1 + \frac{2^{4d}d^{d/2}}{(2^\alpha-1)(d/\alpha-1)}\right)$ is rather pessimistic. More precisely, when points are evenly distributed inside the unit ball the constant is closer to $2^\alpha \cdot \Gamma(1+\alpha/d)$ as $n \to \infty$. Finally, we proved the convergence of the shortest and largest expected weighted length stars inside the unit ball, and we obtained expressions: (i) for their expected weights and (ii) for the expected ratios between them, and also (iii) between them and $EMIT_1(S)$.

6.5   Open Problems

We believe that the results obtained for points uniformly distributed in the unit ball can be extended for non-uniform distribution as well. More precisely, we think that the results obtained for uniform distribution can be extended for any bounded distribution and $0 < \alpha < d$, with a similar technic used by Steele in [212].

Problem 36. *Can the results obtained for points uniformly distributed in the unit ball (Section 6.3.2 and 6.3.3) be extended for non-uniform distribution?*

Finally, the case where $\alpha < 0$ is also important and should be explored in future works.

Problem 37. *Can the results obtained for points uniformly distributed in the unit ball (Section 6.3.2 and 6.3.3) be extended for $\alpha < 0$?*

Chapter 7

State of the Art: Point Location

"Probleme kann man niemals mit derselben Denkweise lösen, durch die sie entstanden sind." — Albert Einstein.[1]

Point location in spatial subdivision is one of the most classical problems in computational geometry [126]. Given a query point q and a partition of the d-dimensional space in regions, the problem is to retrieve the region containing q.

In two dimensions, locating a point has been solved in optimal $O(n)$ space and $O(\log n)$ worst-case query time more then a quarter of a century ago both in theory [159, 160] and in practice [148]. While $O(\log n)$ is the best worst-case query time one can guarantee, it turns out that it is still possible to improve the query time in the average case 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 [146, 64]) 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 [151, 140] two ways to take the spatial coherence into account in point location algorithms: (i) using the *entropy* of the query distribution [32, 142], and (ii) designing algorithms that have a self-adapting capability, i.e., algorithms that are *distribution-sensitive* [143, 91].

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.* [32] or Iacono [142], 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 [163]. However, in many applications the distribution is unknown. Moreover, the distribution of query points can deliberately

---

[1]No problem can be solved from the same level of consciousness that created it.

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 [143] and Demaine *et al.* [91]. 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 $O(n \log n)$.

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 cell to another using the neighborhood relationships between cells, is a straightforward algorithm which does not need any additional data structure besides the triangulation [97]. Walking performs well in practice for Delaunay triangulations, but has a non-optimal complexity [100].

Building on walk. Building on the simplicity of the walk, both the Jump & Walk [168] and the Delaunay hierarchy [93] 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 to reduce the number of visited simplices. In particular, the Delaunay hierarchy guarantees a randomized expected $O(\log n)$ worst-case query time for any Delaunay triangulation in the plane. 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 expected $O(\log n)$ worst-case query time even for Delaunay triangulations in the d-dimensional space. Delaunay hierarchy is currently implemented as the `Fast_location` policy of CGAL [64, 223, 179].

In this chapter, we briefly describe some tiny but representative sampling of all the point location algorithms since 1973, with Knuth's "post-office" problem [152], up to now. We emphasize that this is not intended to be an exhaustive listing of all the existing point location algorithms. (A comprehensive exposition of point location can be found in [210].) The aim is to give enough background for Chapter 8.

7.1   Optimal Planar Point Location

During the 70's and the beginning of the 80's, one of the most studied questions in computational geometry was:

> *"Is it possible to solve the planar point location problem with $O(n)$ space, $O(n)$ preprocessing time, and $O(\log n)$ query time?"*

This problem has been solved in 2D. The quest for an answer has an interesting story, which we succinctly describe here. (The three-dimensional version of this problem is still an open problem [90].)

To not go very far in the past, let us landmark the "post-office" problem, once mentioned by Knuth [152] in 1973, as the starting point of this story. Assume that a set of post-offices are distributed in the plane, then given the location of a residence (in the plane), which of the post-offices is the nearest to that residence? This problem is also known as the *nearest neighbor searching* problem. Dobkin and Lipton [103], Shamos [196]

and Dewdney [102], between 1975 and 1977, showed the relationship between this problem and the Voronoi diagram; i.e., solving the "post-office" problem is equivalent to solving the following problem: Given a query point q, find the polygon of the Voronoi diagram of the input points containing q. Which is basically an *easier* instance of the planar point location problem.

The problem of computing a Voronoi diagram with $O(n)$ space and $O(n \log n)$ time was already known in 1975 [197], also general subdivision search was already known to be efficiently reduced to triangular subdivision (triangulation) search [123]. Great part of the puzzle was solved. Also some close-to-optimal solutions were known: $O(\log n)$ query time with a non optimal memory complexity [103], or $O(\log^2 n)$ query time and $O(n)$ memory complexity [196]. However, the first affirmative answer for the optimal planar point location problem was due to Lipton and Tarjan [159] in 1977, as an application of their *planar separator theorem* [159, 160]. The first solution with an implementation is due to Kirkpatrick [148] in 1983.[2]

Kirkpatrick's Hierarchy. This is the first theoretically optimal and reasonably practical solution of the planar point location problem. As mentioned, a general planar point location problem can be reduced to a point location problem in a triangulation within $O(n \log n)$ time [123] (or even in linear time [65]). Let $\mathcal{T}$ be an arbitrary triangulation with n vertices. A triangulation hierarchy associated with $\mathcal{T}$ is a sequence $\mathcal{T}_1, \ldots, \mathcal{T}_h$ of triangulations, where $\mathcal{T}_1 = \mathcal{T}$ and each region $\mathcal{R}$ of $\mathcal{T}_{i+1}$ is linked to each region $\mathcal{R}'$ of $\mathcal{T}_i$ such that $\mathcal{R}' \cap \mathcal{R} \neq \emptyset$, for $1 \leq i < h$.

Kirkpatrick proved that there exist constants c_1 and c_2 such that for any triangulation $\mathcal{T}$ with n vertices, a triangulation $\mathcal{T}'$ can be constructed in $O(n)$ time, satisfying: (i) the number of vertices in $\mathcal{T}'$ is less or equal to $n \cdot (1 - 1/c_1)$; (ii) each triangle of $\mathcal{T}'$ overlaps at most c_2 triangles in $\mathcal{T}$. Therefore, it is possible to produce a triangulation hierarchy with $h = O(\log n)$ and $O(n)$ space, such that, for any query point q, at most $O(1)$ operations per level in the hierarchy are necessary to find the triangle containing q; this is summarized in Theorem 38 and illustrated in Figure 7.1.

Theorem 38 (Kirkpatrick [148])**.** *There is an $O(\log n)$ search time, $O(n)$ space and $O(n)$ preprocessing time algorithm for the triangular subdivision search problem.*

(a) level 1   (b) level 2   (c) level 3   (d) level 4   (e) level 5
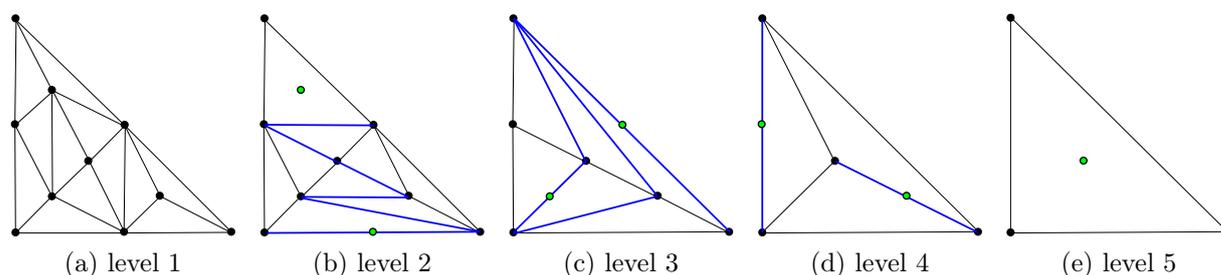
Figure 7.1: **Kirkpatrick's hierarchy.** *An example with five levels of the Kirkpatrick's hierarchy; the first level is $\mathcal{T}$ (left), and the last level is made of one single triangle (right). The green points are the vertices removed from $\mathcal{T}_{i-1}$ to form $\mathcal{T}_i$, and the blue edges are the new edges of $\mathcal{T}_i$.*

---

[2][148] was published in 1983, but Kirkpatrick actually published a technical report two years before.

Other elegant but not so simple solutions came later in 1986 [191, 113, 77], e.g., with an application of the so-called *persistent search trees* of Sarnak and Tarjan [191]. But Kirkpatrick's solution put an end on what we can call: "the first steps of the planar point location problem". Simple and practical solutions for some special subdivisions such as the Delaunay triangulation of a set of points following some particular distributions were already known before 1986 [164]. However, substantially simple and practical solutions for the optimal point location problem for any subdivision in the plane came later in the beginning of the 90's, and are based on randomization.[3] The first randomized point location algorithms are due to Mulmuley [170], Seidel [195], and Boissonnat et al. [50] with a *trapezoidal map* of the subdivision edges. The method is based on inserting the line segments of the subdivision on a random order and maintaining a trapezoidal map of these segments; see Figure 7.2. The point location data structure that results is simply a directed acyclic graph, denoted by *history graph*, that records the history of the various changes to the structure. For a fixed query point, the expected search involves at most $5 \log n + O(1)$ comparisons in expectation; the expectation is taken over all random permutations of the segments. The lowest constant factor of the log expression of the query time was found by Adamy and Seidel [14], who showed that point location queries can be answered in $\log_2 n + 2\sqrt{\log_2 n} + o\left(\sqrt{\log_2 n}\right)$.
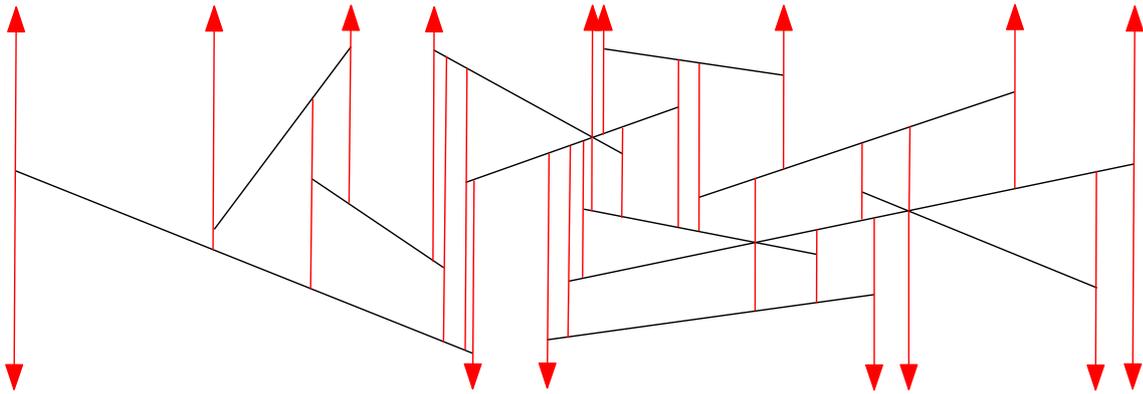
Figure 7.2: **Trapezoidal map.** *Trapezoidal map of several segments.*

7.2 Better Than Worst-Case Optimal

In Section 7.1, we mentioned several algorithms that are worst-case optimal for the planar point location problem. It turns out though, that it is still possible to improve the complexity taking into account the query distribution. Two approaches have been successful in this task: (i) entropy-based point location, (ii) distribution-sensitive point location. We slightly detail each one in what follows.

7.2.1 Entropy-based point location

Suppose we are given a planar subdivision $\mathcal{K}$ and a probability distribution on the set of queries. More precisely, assume that for each cell $s \in \mathcal{K}$, $p_s = Prob\left(q \in s\right)$ is the

---

[3]Clarkson and Shor [76] pioneered the use of randomization in computation geometry.

probability that a query point lies in s.[4] Then the *entropy* of $\mathcal{K}$, denoted by H, is defined as:

$$entropy(\mathcal{K}) = H = \sum_{s \in \mathcal{K}} p_s \log(1/p_s). \tag{7.1}$$

It is possible to design algorithms depending on H instead of n. And H might be $o(\log n)$, e.g., H is $O(1)$ if query points are located in a constant number of cells. However, for the easier one-dimensional point location problem, it is well-known that any algorithm based on comparisons cannot do better than H;[5] see Knuth [152] or Shannon [198].

For subdivisions composed of polygons of constant number of sides (such as a triangulation), the scientific community has produced several algorithms whose time bounds depend only on H. Arya et al. [31] show a data structure with $O(n \log^* n)$ space, which answers query in $H + O(H^{2/3} + 1)$ expected time. This data structure was improved later [33] to $O(n)$ space and $H + O(H^{1/2} + 1)$ expected query time. Other data structures achieving $O(H)$ query time, exist: e.g Iacono [141, 142]. All these algorithms are rather involved. Finally in 2007, Arya et al. presented a randomized data structure, which is reasonably simple and answers a query in expected $(5\ln2)H + O(1)$ and $O(n)$ space, using a *weighted trapezoidal map* [32]; this is summarized in Theorem 39. A weighted trapezoidal map is basically a trapezoidal map that inserts the segments in a specific order depending on the probability that a query lies on a given cell, instead of the classical random order; intuitively, the segments that bound cells of high probability should be added early in the process, since then any query that falls within this cell will have its location resolved near the root of the history graph.

Theorem 39 (Arya et al [32]). *Consider a polygonal subdivision $\mathcal{K}$ of size n, consisting of cells with constant number of sides, and a query distribution presented as a weighted assignment to the cells of $\mathcal{K}$. In time $O(n \log n)$ it is possible to construct a structure of space $O(n)$ that can answer point-location queries in expected time $O(H + 1)$. If $\mathcal{S}$ is presented as a trapezoidal map, then the expected search time is $(5\ln2)H + O(1)$. All bounds hold in expectation over random choices made by the algorithm.*

For subdivisions composed of polygons of variable number of sides, there is no data structure with better complexity than the worst-case optimal. Arya et al. [33] show that even for the restricted case in which the subdivision consists of a single convex n-gon, there exists a query distribution such that no point-location algorithm based on the orientation predicate[6] can achieve an expected query time that is solely a function of entropy.

7.2.2   Distribution-sensitive point location.

Entropy-based point location assumes that a distribution on the set of queries is known. However, in many applications this is not true. Moreover, the distribution of query points can deliberately change over time. Still, we want to have a better complexity than the worst-case optimal if queries are very close to each other. A point location algorithm which adapts to the distribution of the query is called a *distribution-sensitive* point location algorithm, or *self-adapting* point location algorithm.

---

[4]Analyses of entropy-based point location presented in this section assume that the accesses to the cells are independent one each other; this is sometimes too restrictive in practice.

[5]Actually, a better lower bound of $H + \Omega\left(\sqrt{H}\right)$ for planar point location was recently discovered [163].

[6]In other words, discovering in which side of a line a given point lies in.

Previously, we saw that given the access probability of each cell of a triangulation (assumed to be independent), it is possible to create a data structure whose expected query time is the entropy of that probability distribution. Such a result is analogous to the one-dimensional structure known as *optimal binary search tree* [151]. On the other hand, *splay trees* have the same amortized asymptotic query time as optimal search trees [140], without having to know the access probabilities. The question is: "Is there an analogous of the splay trees for point location in the plane?"

To the best of our knowledge, just a few results on distribution-sensitive point location exist up to 2010: Demaine *et al* [91], or Iacono and Langerman [143] data structures. They are rather involved, requiring more than linear memory, and more than $O(n \log n)$ pre-processing time. The complexity of their approach is in term of some distances, say *dist*, between two successive queries q_{i-1}, q_i. Naturally, *dist* should depend on the triangulation. And for general planar triangulations and general distance metrics, it is not possible to produce a data structure with a better complexity than the worst-case optimal [91]; mainly because it is possible to have, for a point p, 2^i points q such that $dist(p, q) = i$, such phenomenon disallows the time complexity to be $o(\log n)$.

Iacono and Langerman [143] data structure is the most successful distribution-sensitive point location data structure in terms of pre-processing and memory complexy, up to 2010, to the best of our knowledge. It uses the β-diamond metric ($\beta\Diamond$). The $\beta\Diamond$-distance, denoted by $d_{\widehat{\beta\Diamond}}(p, q)$, is defined as follows: Let R_β be the diamond shaped region with vertices p and q, symmetry axis pq, and angles at p and q both equal to a fixed constant $\beta > 0$. Now, let $\beta\Diamond(p, q)$ be an r-offset of R_β, with r the largest edge of the triangulation intersecting R_β. Then $d_{\widehat{\beta\Diamond}}(p, q)$ is the number of edges of the triangulation intersecting $\beta\Diamond(p, q)$; see Figure 7.3. It is proved that such a metric satisfies the following properties: (i) *monotonicity*, if points p_1, p_2 and p_3 appear in that order on a line, then $d_{\widehat{\beta\Diamond}}(p_1, p_2) \leq d_{\widehat{\beta\Diamond}}(p_1, p_3)$; (ii) *sanity*, for a point q, the number of elements in $\{p \mid d_{\widehat{\beta\Diamond}}(p, q) < k\}$ is polynomial on k. Because of the geometry of $\beta\Diamond(p, q)$, the monotonicity of $d_{\widehat{\beta\Diamond}}$, and the sanity $d_{\widehat{\beta\Diamond}}$, with a clever (but rather involved) usage of pointers, $(2, 4)$-trees [139] and persistent search trees [191], Iacono and Langerman [143] proved Theorem 40.

Theorem 40 (Iacono and Langerman [143]). *For any angle $\beta < \pi/2$, it is possible, in $O\left(\frac{1}{\beta} n \log^2 n \log \log n\right)$ time, to construct a data structure of size $O\left(\frac{1}{\beta} n \log \log n\right)$ so that given points p and q and a pointer to the cell containing p, the data structure returns the cell containing q in $O\left(\frac{1}{\beta} \log d_{\widehat{\beta\Diamond}}(p, q)\right)$ time.*

We show in Section 8.4, that if the triangulation is Delaunay and satisfies some hypotheses, than it is possible to construct a data structure with $O(n \log n)$ preprocessing time, $O(n)$ memory, and $O(\log \sharp(q_{i-1} q_i))$ (amortized) expected query time, where $\sharp(pq)$ indicates the expected number of simplices crossed by the line segment pq.

7.3   Practical Methods for Point Location

In the last sections, we presented several optimal approaches to locate a point on a triangulation; in this section we present non-optimal approaches behaving well in practice. The main interests are three: (i) algorithms in previous sections are rather involved, and practitioners may prefer to implement simpler algorithms; (ii) non-optimal algorithms may be
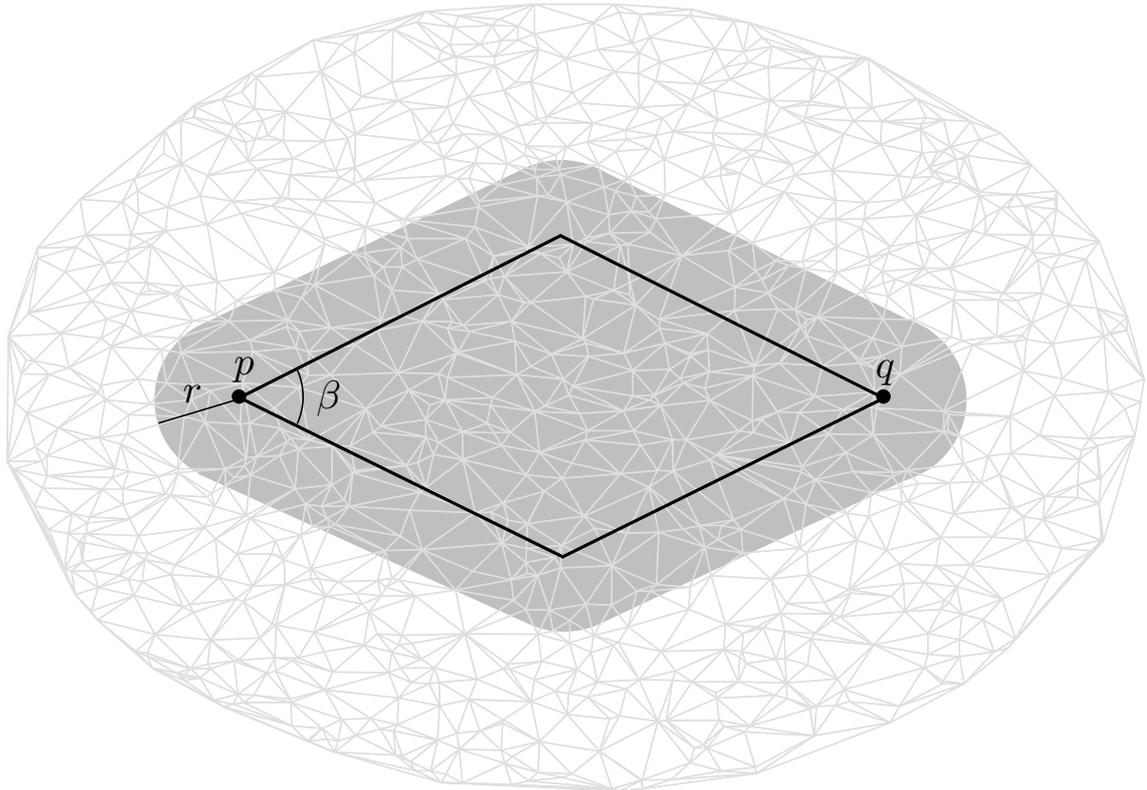
Figure 7.3: β-**diamond metric.** *Diamond shaped region with vertices p and q.*

faster than optimal ones in practice; (iii) the optimal algorithms are for triangulations in the plane, which is highly insufficient for nowadays applications, where three-dimensional triangulations are required. As a plus, one of the algorithm we present here, the Delaunay hierarchy [93], is optimal (in expectation) in two dimensions, and also in higher dimensions under some (not so restrictive) hypotheses.

All the algorithms we present in this section are based on the notion of *walking* in a triangulation; the best ones (Jump & Walk and Delaunay hierarchy) combine walking with a good *starting point*. Assume hereafter that a triangulation $\mathcal{T}$ of n points, a query point q and the location of another point p (the starting point) are given, we present some existing practical methods for point location.

7.3.1   Walk Strategies

Visibility Graph

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 cells of $\mathcal{T}$, and a pair of cells (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 7.4. When two cells σ_i and σ_j are such that $(\sigma_i, \sigma_j) \in E$, we say that σ_j is a *successor* of σ_i. Now, a *visibility walk* consists in repeatedly walking from a cell σ_i to one of its successor in $\mathcal{VG}$ until the cell containing q is found; a *walking strategy* describes how this successor is chosen.
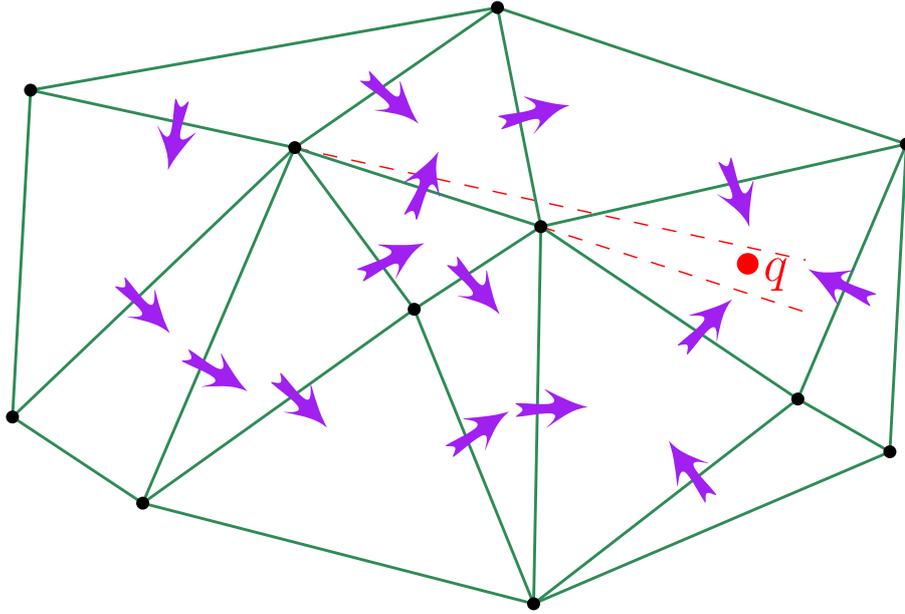
Figure 7.4: **Visibility graph.** *Arrows represent edges of* $\mathcal{VG}$.

Given two adjacent cells, deciding which one is a successor of the other relies on the *orientation predicate* (presented in Section 2.2.4). By convention, the representation of a cell σ of $\mathcal{T}$ gives its vertices $p_0, \dots, p_d$ in an order corresponding to a positive orientation

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

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

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

Straight Walk

Straight walking in a triangulation is almost as old as Lipton and Tarjan solution of the planar point location problem [159], and certainly straightforward. The main idea comes from the early papers on computational geometry [155, 127, 55]. It relies on the fact that the internal representation of $\mathcal{T}$ gives a constant-time access between neighboring cells (which is the case of CGAL's representation [179]; details are given in Section 2.2.2). Now, consider the line segment $s = pq$, the procedure to locate a query point q is to walk through all the cells intersected by s; this is done by repeatedly walking from a cell σ to the successor in $\mathcal{VG}$ of σ that intersects s. The walk starts at a cell containing p and stops at the cell containing q. This procedure is classically called *straight walk*; straight walk has a worst-case complexity linear in the number of cells of the triangulation [203], see Figure 7.5.
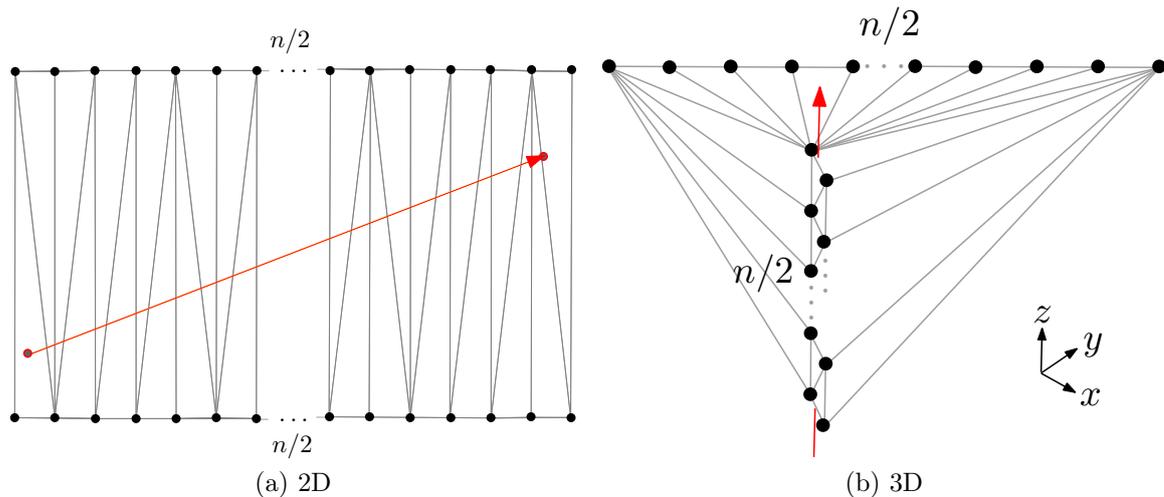
Figure 7.5: **Worst-case straight walk.** *(a) stabbing $\Omega(n)$ cells in the plane; (b) stabbing $\Omega(n^2)$ cells in the space.*

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 cells stabbed by s is proportional to $n^{1/d}$ [127, 55], or even more precisely

$$O\left(\|s\| \cdot n^{1/d}\right). \tag{7.2}$$

The term $\|s\|$ when used inside a Big-O notation, rigorously speaking, represents a ratio between two quantities having the same unity of measure. Here is the shortcut: $\|s\|$ abbreviates $\|s\|/\delta$ (when it applies), where δ is the diameter of the domain boundary. In this thesis, as the domain is bounded in all the theorems where this notion is relevant (independently of n), then we took the liberty to omit δ, as it is a constant. And hence, we simplify $\|s\|/\delta$ to $O\left(\|s\|\right)$.

Boundaries make the analysis significantly more complex. In the plane, Bose and Devroye [53] shows that s being close to the boundary[7] does not affect Eq.(7.2) in expectation.[8] Actually the distribution does not even need to be uniform, see Theorem 41.

Theorem 41 (Bose and Devroye [53])**.** *Let $0 < \alpha \leq f(x,y) \leq \beta < \infty$ be a probability distribution of points with a convex compact support $\mathcal{C} \subset \mathbb{R}^2$, for fixed constants α, β. Let s be a fixed line segment contained in $\mathcal{C}$. Then the expected number of intersections between s and the Delaunay triangulation of n random points following the distribution f is $O\left(\|s\| \cdot n^{1/2}\right)$.*

In three dimensions, there is no similar bound as in Theorem 41. The best bound is due to Mücke et al. [168] and its underlying probabilistic analysis is rather technical. They basically showed that, under some slightly more restrictive hypotheses, the number of intersected Delaunay cells is proportional to $O\left(n^{1/3} \log n / \log \log n\right)$; see Theorem 42.

---

[7]It is assumed in [53] that the boundary contains at least one circle of positive radius.

[8]The expected worst-case complexity has been shown lately to be $O(\sqrt{n})$ [52] as well.

Theorem 42 (Mücke et al. [168]). *Let $0 < \alpha \leq f(x, y, z) \leq \beta < \infty$ be a probability distribution of points with a convex compact support $\mathcal{C} \subset \mathbb{R}^3$ having small curvature, for fixed constants α, β. Let s be a fixed line segment inside $\mathcal{C}$, but at a distance of at least $\Omega\left((\log n/n)^{1/3}\right)$ from the boundary of $\mathcal{C}$. Then the expected number of intersections between s and the Delaunay triangulation of n random points following the distribution f is $O\left(n^{1/3}\log n / \log \log n\right)$.*

It is conjectured though, that even considering boundaries, for n evenly distributed points inside a convex domain of $\mathbb{R}^d$, the actual expected total number of intersected cells of the Delaunay triangulation of those points is $O\left(\|s\| \cdot n^{1/d}\right)$.

Stochastic Walk

Let q be the query point, and σ be a cell of $\mathcal{T}$, with vertices on points $p_0, \ldots, p_d$. Then recall that an adjacent cell σ' is the successor of σ in $\mathcal{VG}(\mathcal{T}, q)$ if the supporting plane of their common facet (say the facet containing the points $p_1, \ldots, p_d$) separates p_0 and q; this is true if $orient(q, p_1, \ldots, p_d) = -1$.

Before describing the stochastic walk procedure, let us remind a trick to handle queries that are not inside the convex hull of $\mathcal{T}$ (also presented in Section 2.2.4). It consists into compactifying $\mathbb{R}^d$ into the topological sphere by: (i) adding a point at infinity ∞, and then (ii) gluing ∞ and each facet of the convex hull of $\mathcal{T}$, forming new simplices (one for each facet). These new simplices are called *infinite simplices*. When an infinite simplex has dimension d, it is called an *infinite cell*.

We describe now the fundamental procedure of the stochastic walk algorithm, which runs for each visited cell. Assume σ is the current cell being visited, and i is a random index chosen from $0 \ldots d$.

Selecting next cell to visit. The adjacent cells of σ $(\sigma_0, \sigma_1, \ldots, \sigma_d)$ are tested sequentially starting at σ_i (i random) to be a successor of σ. The first cell, which is a successor of σ in the sequence, is selected.

Cell found. When there is no successor of σ, then σ contains q, and the remaining work is to count the number of orientation tests that result in 0 in order to know exactly whether q is inside σ or on its boundary (and which piece of boundary).

Cell not found. If σ is an infinite cell, then q is outside the convex hull of $\mathcal{T}$.

A straightforward modification of the stochastic walk leads to a more efficient procedure.[9] It consists simply into not selecting the previous visited cell as a candidate to be the next cell to visit [97]. This effectively avoids one orientation predicate computation. To implement this modification one needs only to remember the previous visited cell during the walk and compare it with σ_j. CGAL incorporates this modification in its stochastic walk implementation, and so do we in the next chapter.

Current results on the complexity of the stochastic walk are weaker than the results for straight walk: It is known that the stochastic walk finishes with probability 1 [97], though it may visit an exponential number of cells (even in $\mathbb{R}^2$). In the case of Delaunay triangulations, the complexity becomes linear in the number of cells. For evenly distributed points, the $O(\|s\| \cdot n^{1/d})$ complexity is also conjectured for stochastic walk, but remains unproved even in the plane; see Figure 7.6. In practice, stochastic walk answers point

---

[9]The modified stochastic walk is often referred to as the *remembering stochastic walk*.

(a) Varying n

(b) Varying $\|s\|$
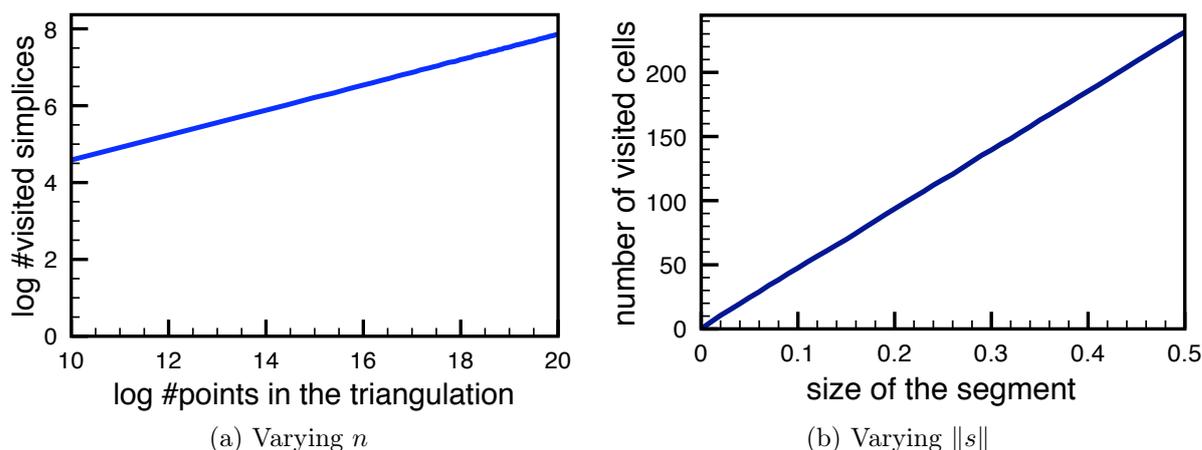
Figure 7.6: **Stochastic walk complexity conjecture.** *(a) shows the logarithm in base 2 of the average number of visited cells by a stochastic walk done from the cell containing the point $(1/2, 1/2, 1/4)$, to the query point $(1/2, 1/2, 3/4)$, in several Delaunay triangulations of $2^{10}, \ldots, 2^{20}$ points evenly distributed in the unit cube. The slope of the curve in this log-graph shows approximately $1/3$, which indicates an approximate $O(n^{1/3})$ visited cells, where n is the number of points. (b) shows the average number of visited cells by a stochastic walk done from the cell containing the point $(1/2, 1/2, 1/2 - l/2)$, to the query point $(1/2, 1/2, 1/2 + l/2)$, for $l = 0.00, 0.01, \ldots, 0.50$, in several Delaunay triangulations of 2^{20} points evenly distributed in the unit cube. Note the linear behavior.*

location queries faster than the straight walk [97] and it is the current choice of CGAL for the walking strategy [223, 179], in both dimensions 2 and 3.

In general. For simplicity, straight walk is used for analyses; stochastic walk is used in practice. Other walk strategies exist,[10] but are not considered in this thesis.

7.3.2 Jump & Walk

Jump & Walk takes a random sample of k vertices of $\mathcal{T}$, called *landmarks*, and uses a two-steps location process to locate a query q. First, the jump step determines the nearest landmark in the sample in (brute-force) $O(k)$ time, then a walk in $\mathcal{T}$ is performed from that vertex; see Figure 7.7.

Assuming that: (i) $\mathcal{T}$ is Delaunay, (ii) points in $\mathcal{T}$ are evenly distributed in a convex region of the space, and (iii) the query point is far enough from the boundary, then Jump & Walk becomes easy to analyze. Because of the uniformity of the distribution, for each landmark p, an expected $O(n/k)$ points are located inside a ball centered at p. And assuming the aforementioned hypotheses, a walk from a landmark to a query point takes $O\left((n/k)^{1/d}\right)$ time, see Section 7.3.1. By summing up the cost of the brute-force jump step, we have an expected $O\left((n/k)^{1/d} + k\right)$ time complexity, which is optimized taking $k = n^{1/(d+1)}$. This gives a final complexity of $O\left(n^{1/(d+1)}\right)$.

Naturally, if we don't assume the hypotheses above, the analysis becomes harder. Devroye et al. [101] shows that, under similar hypotheses as the ones in Theorem 41, if the query point is independent of the points of the triangulation, then the time to locate

---

[10]For instance, *compass routing* [153] or *orthogonal walk* [97].
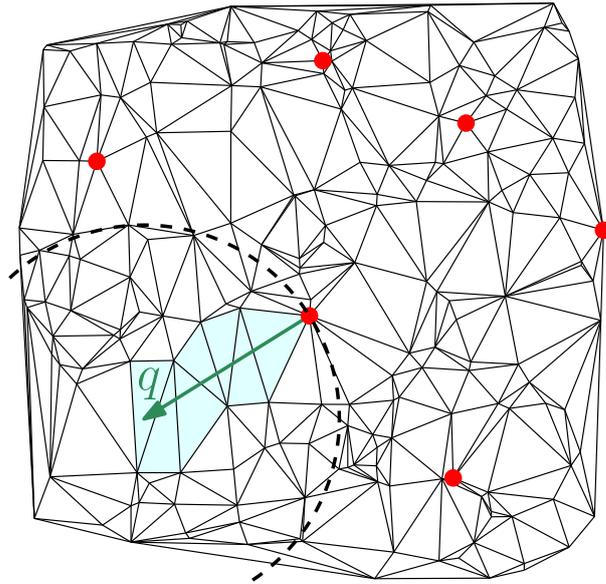
Figure 7.7: **Jump & Walk.** *The walk starts from the nearest landmark (represented by dots above) with respect to the query. Then it ends at the cell containing the query.*

a query point is $O\left(n^{1/3}\right)$; see Theorem 43.

Theorem 43 (Bose and Devroye [101]). *Let $0 < \alpha \leq f(x, y) \leq \beta < \infty$ be a probability distribution of points with a convex compact support $\mathcal{C} \subset \mathbb{R}^2$, for fixed constants α, β. Let S be a set of n random points following the distribution f. Assume that a query point q is independent of the choice of points in S. Then the expected time to locate q in the Delaunay triangulation of S using Jump & Walk is $O\left(n^{1/3}\right)$.*

In Section 8.2.3, we show that this still works in a more general framework, where the random sample can evolve and adapt to the distribution over time.

In three dimensions, there is no similar bound as in Theorem 43, which is understandable, since the best bounds on the size of a walk is not "yet" proved to be $O\left(n^{1/3}\right)$. As an extension of Theorem 42, Mücke et al. [168] shows that, the jump and walk strategy takes $O\left(\delta^{1/4}n^{1/4}(\log n / \log\log n)^{3/4}\right)$ time, where δ is the expected degree of a vertex; see Theorem 44.

Theorem 44 (Mücke et al. [168]). *Let $0 < \alpha \leq f(x, y, z) \leq \beta < \infty$ be a probability distribution of points with a convex compact support $\mathcal{C} \subset \mathbb{R}^3$ having small curvature, for fixed constants α, β. Let S be a set of n random points following the distribution f. Assume that a query point q is independent of the choice of points in S, and is at distance of at least $\Omega\left(n^{-1/18}\right)$ from the boundary of $\mathcal{C}$. Then the expected time to locate q in the Delaunay triangulation of S using Jump & Walk is $O\left(\delta^{1/4}n^{1/4}(\log n / \log\log n)^{3/4}\right)$.*

The expected maximum degree of a vertex is $O(\log n / \log\log n)$ [45], and $O(1)$ [109] if points are chosen uniformly at random in a d-dimensional ball. And hence the expected cost here is bounded by $O\left(n^{1/4} \log n / \log\log n\right)$, and $O\left(n^{1/4}(\log n / \log\log n)^{3/4}\right)$ for points chosen at random in a d-dimensional ball. Nevertheless, the actual expected time complexity is conjectured to be $O\left(n^{1/4}\right)$, even with boundaries.

7.3.3 Delaunay hierarchy

Building on the idea of Jump & Walk, the Delaunay hierarchy [93] uses several levels of random samples: At each level of the hierarchy, the walk is performed starting at the closest vertex of the immediately coarser level. Building the hierarchy by selecting a point in the coarser level with some fixed probability, yields a good complexity. In two dimensions, the complexity is $O(\log n)$ in the worst case. In higher dimensions, this logarithmic time holds if random samples of $\mathcal{T}$ have a linear number of cells.

Given a set of n points S in the plane, the Delaunay hierarchy [93] constructs random samples $S = S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots \subseteq S_h$ such that $Prob(p \in S_{i+1} \,|\, p \in S_i) = 1/\alpha$ for some constant $\alpha > 1$. $\mathcal{DT}_i$, the Delaunay triangulation of S_i, is computed for $0 \leq i \leq h$, 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 $i + 1$.

Devillers [93, Lemma 4] shows 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. The crux of the proof resides in the fact that the expected degree of the nearest neighbor of a vertex in $\mathcal{T}$ is bounded in the plane for any distribution of points. From the size of the samplings described above, the total size of the structure is $O(n) \cdot \sum_{i=0}^{\infty} \alpha^{-i} = O(n)$. Theorem 45 summarizes the behavior of the hierarchy for points in the plane.

Theorem 45 (Devillers [93]). *The construction of the Delaunay hierarchy of a set of n points is done in expected time $O(n \log n)$ and $O(n)$ space, and the expected time to locate a query q is $O(\log n)$. The expectation is on the randomized sampling and order of insertion, with no assumption on point distribution.*

Theorem 45 holds for Delaunay triangulation $\mathcal{T}$ of arbitrary finite dimension d, as long as the number of cells is $O(n)$, and the expected degree of the nearest neighbor of a vertex in $\mathcal{T}$ is bounded. Since the *kissing number* [177] in any finite dimension d is bounded, the condition holds when the expected degree of vertices in each random sample of $\mathcal{T}$ is $O(1)$. And hence, $\mathcal{T}$ having random samples of linear size, is a sufficient condition for Theorem 45 to hold in higher dimensions [93].

Delaunay hierarchy supports insertion and deletion of vertices; i.e., it is dynamic, while being optimal. Another interesting feature of the Delaunay hierarchy is that the average number of visited simplices during the walk on a particular level of the triangulation tends to be similar for each level when n is big enough. Figure 7.8 shows the distribution of the number of simplices visited during the stochastic walk done on the last level (level h) of the hierarchy for some set of points. Figure 7.9 shows experimentally the impact of the number of vertices on the distribution.

If query points are given one at a time, Delaunay hierarchy is one of the best solution in practice for two and three dimensions. The Delaunay hierarchy, for points in two and three dimensions, is implemented in CGAL [223, 179].

We show in Section 8.4, that, under some hypotheses on the triangulation, the Delaunay hierarchy can be used to achieve very fast distribution-sensitive point location as well.

(a) Square

(b) Ellipse

(c) Cube

(d) Ellipsoid
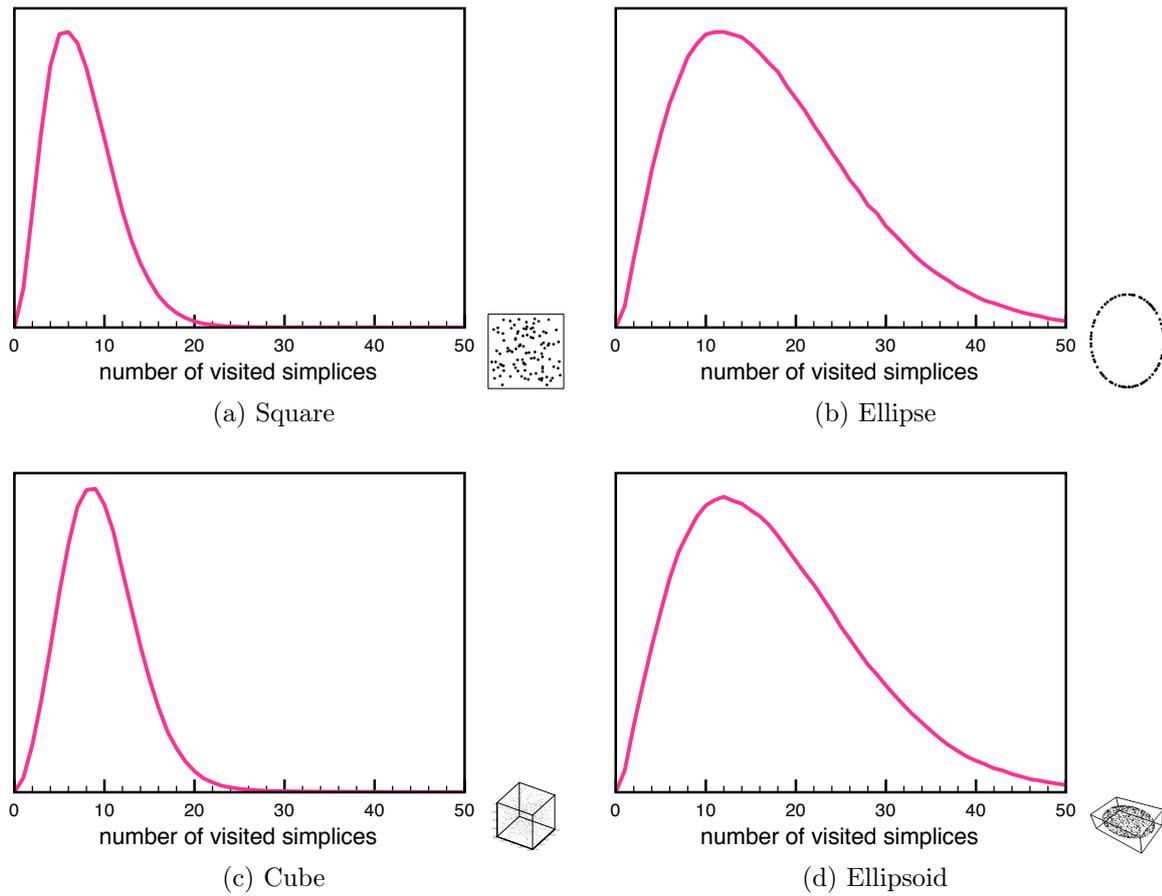
Figure 7.8: **Delaunay hierarchy with stochastic walk.** *The distribution of the number of simplices visited during the stochastic walk done on the last level of the hierarchy for 2^{20} points: (a) evenly distributed in a square, (b) evenly distributed on an ellipse, (c) evenly distributed in a cube, (d) evenly distributed on an ellipsoid.*
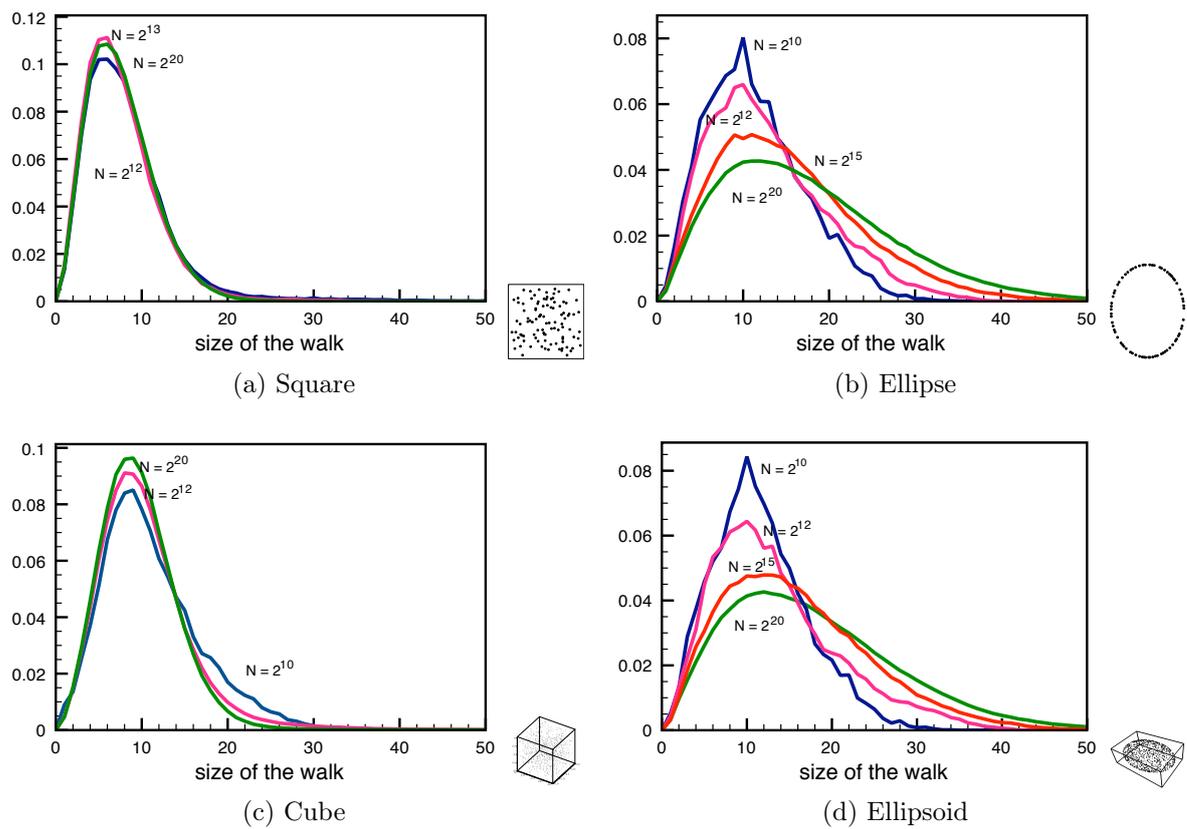
Figure 7.9: **Varying the number of vertices.** *The distribution of the number of simplices visited during the stochastic walk done on the last level of the hierarchy for various number of random points: (a) evenly distributed in a square, and (b) evenly distributed on an ellipse, (c) evenly distributed in a cube, and (d) evenly distributed on an ellipsoid.*

Chapter 8

Simple and Efficient Distribution-Sensitive Point Location, in Triangulations

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better." — Edsger Wybe Dijkstra.

- P. M. M. de Castro and O. Devillers. Practical Self-Adapting Point Location, in Triangulations. Submitted, 2010. (Also available as: Research Report 7132, INRIA, 2010.)

- P. M. M. de Castro and O. Devillers. Simple and Efficient Distribution-Sensitive Point Location, in Triangulations. Submitted, 2010

Point location in spatial subdivision is one of the most studied problems in computational geometry. In the case of triangulations of $\mathbb{R}^d$, we revisit the problem to exploit a possible coherence between the query points. We propose several new ideas to improve point location in practice. Under some hypotheses verified by "real point sets", we also obtain interesting theoretical analysis. Chapter 6 and 7 present most of the fundamentals used in this chapter.

Our Contributions. In Section 8.1, 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 8.5.1, we provide experimental evidence that some realistic triangulations verify 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 $\mathbb{R}^d$ in order to obtain complexity results. Results on the length of such trees can be reviewed in Chapter 6.

In Section 8.2, we investigate constant-size-memory strategies to choose the starting point of a walk. More precisely, we compare strategies that are dependent on previous queries (self-adapting strategies) and strategies that are not (non-self-adapting strategies), mainly in the case of random queries. Random queries are *a priori* not favorable to self-adapting strategies, since there is no coherence between the queries. Nevertheless, our computations prove that self-adapting strategies are, either better, or not really worse in this case. Thus, there is a good reason for the use of self-adapting strategies since they are competitive even in situations that are seemingly unfavorable. Section 8.5.2 provides experiments to confirm such behavior on realistic data.

In Section 8.3, 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 [134] that similar ideas in the plane give 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 expected complexity of the classical Jump & Walk. We also provide analysis for some modified versions of Keep, Jump, & Walk. In Section 8.5.3, 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 8.4, we show that *climbing* the Delaunay hierarchy can be used to answer a query q in $O(\log \sharp(pq))$ randomized expected complexity, where p is a point with a known location and $\sharp(s)$ indicates the expected number of cells crossed by the line segment s. Climbing instead of walking turns Keep, Jump, & Walk into Keep, Jump, & Climb, which appears to take the best of all methods both in theory and in practice.

8.1 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.** *Given a triangulation scheme (such as Delaunay, Regular, ...), and a distribution of points with compact support $\Sigma \subset \mathbb{R}^d$, $\mathcal{C} \subseteq \Sigma$: For a triangulation $\mathcal{T}$ of n points following the given distribution and built upon the given triangulation scheme, the Distribution Condition is verified if there exists a constant $\kappa \in \mathbb{R}$, and 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** $1 + \kappa \cdot \|s\| \cdot \mathcal{F}(n)$.*

One known case where the Distribution Condition is verified is the Delaunay triangulations of points following the Poisson distribution in dimension d, where $\mathcal{F}(n) = O(n^{1/d})$, for any region $\mathcal{C}$; see Figure 8.1-left. 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 of dimension δ embedded in dimension d. We claim that the relevant dimension is in fact the one of the manifold (see Figure 8.1-right), this claim is supported by our experiments (Section 8.5.1) and stated in the following conjecture:
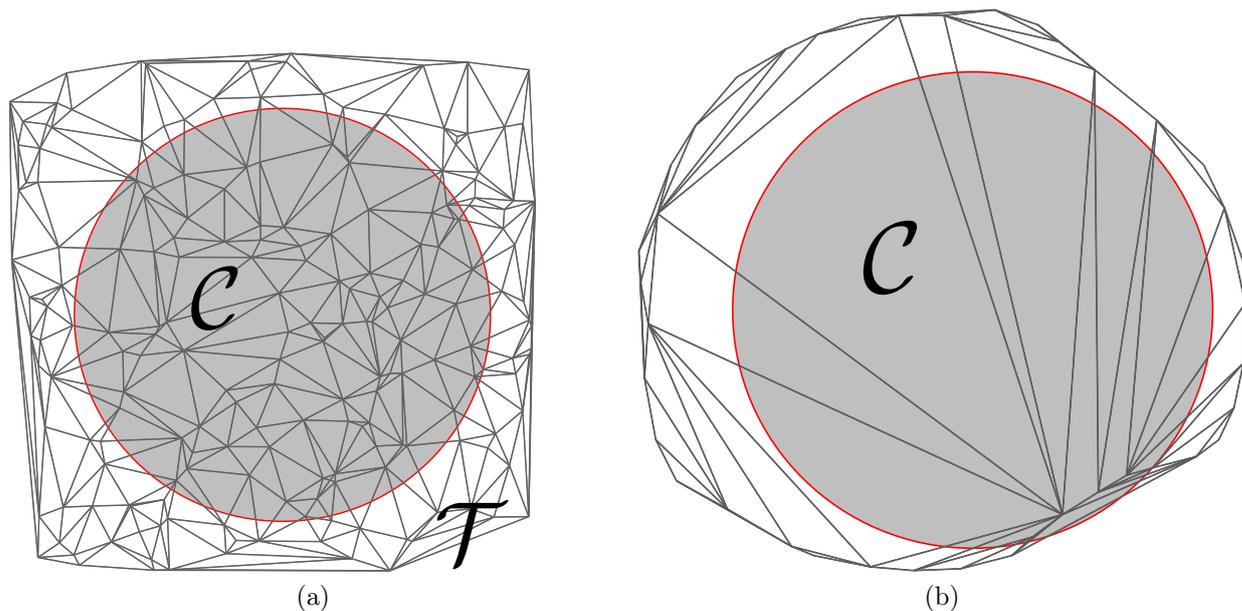
Figure 8.1: **Distribution Condition.** *(a)* $\mathcal{F}(n) = O(\sqrt{n})$, *(b)* $\mathcal{F}(n) = O(n)$.

Conjecture 46. *The Delaunay triangulations in dimension d of n points evenly distributed on a bounded manifold Π of dimension δ, verify the Distribution Condition inside the convex hull of Π, with $\mathcal{F}(n) = O(n^{1/\delta})$.*

The Distribution Condition ensures 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 in $\mathbb{R}^d$, if the Distribution Condition is verified 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 $\mathcal{C}$ is at most

$$\kappa \cdot \mathcal{F}(n) \cdot \sum_{i=1}^{m} |e_i| + m, \tag{8.1}$$

where e_i is the line segment formed by the i-th starting point and the i-th query point.

Now, please take a look at the expression $\sum_{i=1}^{m} |e_i|$ above. 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 cell visited during a previous walk. Thus the k segments e_i, $1 \leq i \leq m$, 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.(8.1) can be rewritten as the following expression:

$$\kappa \cdot \mathcal{F}(n) \cdot \|\mathcal{E}\| + m. \tag{8.2}$$

8.2   Constant-Size-Memory Strategies

In this section, we analyze the Location Tree length of strategies that store a constant number of possible starting points for a straight walk. We also provide a comparative study between them.

8.2.1   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 *point at infinity*, detailed in Section 2.2.2; 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 47. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside the unit ball, then the expected length of the Location Tree of the best fixed-point strategy is*

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

Proof. Restricted case of Theorem 31 for $\alpha = 1$.   □

Theorem 48. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside 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 Beta function.

Proof. Restricted case of Theorem 32 for $\alpha = 1$.   □

Corollary 49. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside 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 8.2 gives the expected average length of an edge of the best and worst fixed-point Location Trees.

8.2.2   Last-point strategy.

An easy 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 $\mathcal{T}$ verifies the Distribution Condition, the optimal path is the $EMLP(S)$. (Definition of $EMLP(S)$ can be found in Section 6.1.)

In the worst case, the length of such a path is clearly $\Omega(m)$; an easy 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 [214, 121]. Furthermore, in two dimensions, one could find such permutation in $O(m \log m)$ time complexity [181].

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

Theorem 50. *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 $S = p_1, \dots, p_m$, and any fixed-point c. Then we have:

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

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

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

which completes the proof.   □

Theorem 51. *The ratio between the lengths of the Location Tree of the last-point strategy and the fixed-point strategy can be 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 50 and Theorem 51, 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 52. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside 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 [190]. □

Theorems 47, 48, and 52 lead to the following corollary:

Corollary 53. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside 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 Location Tree lengths of the last-point and the worst fixed-point strategies is $2d/(2d+1)$.*
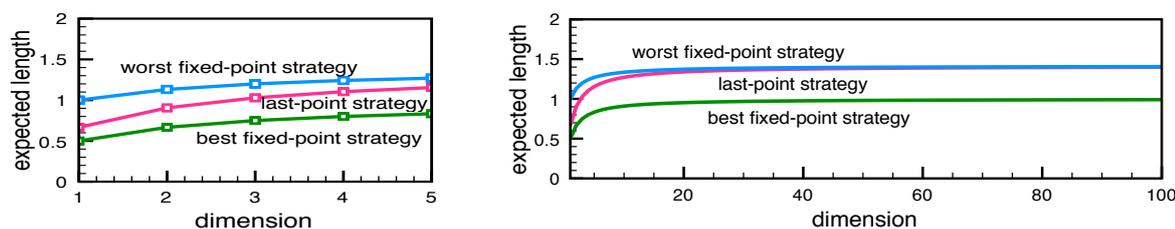
Figure 8.2: **Expected lengths.** *Expected average lengths of an edge of the last-point, best and worst fixed-point Location Trees. The domain $\mathcal{C}$ for each dimension d is the d-dimensional unit ball, and the queries are evenly distributed in $\mathcal{C}$.*

As shown in Figure 8.2, 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.

Now, one might ask whether the shape of $\mathcal{C}$ affects the cost of the strategies. In the quest for an answer, we may consider query points independent and identically distributed following the uniform distribution inside the the unit cube $[0,1]^d$. This leads to the following related expressions:

$$B_d = \int_0^1 \ldots \int_0^1 \left(\lambda_1^2 + \ldots + \lambda_d^2\right)^{1/2} \mathrm{d}\lambda_1 \ldots \mathrm{d}\lambda_d,$$

$$X_d = \int_0^1 \ldots \int_0^1 \left((\lambda_1 - 1/2)^2 + \ldots + (\lambda_d - 1/2)^2\right)^{1/2} \mathrm{d}\lambda_1 \ldots \mathrm{d}\lambda_d,$$

$$\Delta_d = \int_0^1 \ldots \int_0^1 \left((\lambda_1 - \lambda_1')^2 + \ldots + (\lambda_d - \lambda_d')^2\right)^{1/2} \mathrm{d}\lambda_1 \ldots \mathrm{d}\lambda_d \, \mathrm{d}\lambda_1' \ldots \mathrm{d}\lambda_d',$$

where B_d, X_d, and Δ_d are respectively the average length of an edge of: the largest star (rooted at a corner), the smallest star (rooted at the center), and of a random path. Above expressions are often referred to as *box integrals* [38]. First, note that by substitution of variable we have $B_d/X_d = 2$, independently of d. In Anderssen *et al.* [183], we have that, $B_d \sim \sqrt{d/3}$ and $\Delta_d \sim \sqrt{d/6}$ and thus B_d/Δ_d and $\Delta_d/X_d \sim \sqrt{2}$ when d goes to infinity.

These ratios have to be compared with Corollary 53. If $\mathcal{C}$ is an unit cube, the expected Location Tree length of the last-point, best and worst fixed-point strategies remains in bounded ratio, but with different values compared to the case where $\mathcal{C}$ is a ball. Notice however that, when d is large, the ratio between the best fixed-point and the last-point strategies remains $\sqrt{2}$ in both cases.

8.2.3 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 landmarks 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 $EMIT_k(S)$. It has bounded degree $k+1$ (or the *kissing number* $+1$ in dimension d, if it is smaller than $k+1$) and its length is greater than $\|EMST(S)\|$ and smaller than the length of the path obtained by visiting the vertices in the same order, thus previous results provide upper and lower bounds. (Definitions of $EMIT_k(S)$ and $EMST(S)$ can be found in Section 6.1.)

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. Theorem 22 (for $\alpha = 1$) bounds the size of the Location Tree of the k-last-points strategy for k successive queries, though the constant $\gamma_{d,1}$ seems too big. However this constant is rather pessimistic if we refer to practical data sets; Theorem 54 leads to Corollary 55, which gives a better constant for queries evenly distributed inside the unit sphere. (This observation can be found in Section 6.3.2 within a more general context.)

Theorem 54. *Let S be a sequence of m query points independent and identically distributed following the uniform distribution inside the unit ball, then the expected length of the Location Tree of the k-last-points strategy verifies*

$$\left(\frac{1}{d}\right) B\left(k+1, \frac{1}{d}\right) \cdot m \le E(length) \le 2 \left(\frac{1}{d}\right) B\left(k+1, \frac{1}{d}\right) \cdot m. \qquad (8.3)$$

Proof. Restricted case of Theorem 26 for $\alpha = 1$. $\qquad \square$

Corollary 55. *Let S be a sequence of k query points independent and identically distributed following the uniform distribution inside 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 54, 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 $l_i \le \left(\frac{2}{d}\right) B\left(i+1, \frac{1}{d}\right)$. 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 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. $\qquad \square$

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 8.5 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, $k = 4$ improves the location time by up to 15% on some data sets.

8.3   Keep, Jump & Walk

8.3.1   Jump & Walk

The Jump & Walk technique takes a random sample of k vertices of $\mathcal{T}$, and uses a two-steps location process 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. The usual analysis of Jump & Walk makes the hypothesis that $\mathcal{T}$ is the Delaunay triangulation of points evenly distributed. Taking $k = n^{1/(d+1)}$ gives a complexity of $O(n^{1/(d+1)})$ [168, 101]. More details can be found in Section 7.3.2.

8.3.2   A Simple Modification: Distribution Sensitiveness

The classical Jump & Walk strategy [168, 101] uses a set of k landmarks randomly chosen in the vertices of $\mathcal{T}$; a query is located by walking from the closest amongst the landmarks. In order to ensure the distribution sensitiveness of such a strategy, 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.[1]

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, and
— Walk: the time $W(k)$ to walk from the closest landmark to q.

This modification performs surprisingly well in practice, experimental results for method (ii) are provided in Section 8.5.3.

In this section, we analyze such a strategy. The analysis consider the **straight walk** as the walk strategy. We start with the following lemma, which is a fundamental piece of the analysis.

Lemma 56. *Let $\mathcal{T}$ be a triangulation of n points following some distribution with compact support in $\mathbb{R}^d$, if the Distribution Condition is verified 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. (Definition of $EMIT(S)$ can be found in Section 6.1.) Let $\mathcal{T}$ be a triangulation of n points following some distribution with compact support in $\mathbb{R}^d$, if the Distribution

---

[1]The strategy presented in this section is similar to the k-last-points strategy. The main difference is that k is not necessarily a constant anymore, and hence it is allowed to be $\omega(1)$; details on the k-last-points strategy can be found in Section 8.2.3.

Condition is verified 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.(8.2), at most

$$\kappa \cdot \mathcal{F}(n) \cdot \sum_{e \in \mathcal{E}} \|e\| + k = \kappa \cdot \mathcal{F}(n) \cdot \|\mathcal{E}\| + k = O\left(\mathcal{F}(n) \cdot k^{1-1/d} + k\right), \qquad (8.4)$$

since, by Theorem 22 (for $\alpha = 1$), $\|EMIT\| = O(k^{1-1/d})$. Therefore, $W(k)$ has an expected amortized $O\left(\mathcal{F}(n) \cdot k^{-1/d} + 1\right)$ complexity for k queries. (Recall from Section 8.1 that the Distribution Condition does not force $\mathcal{T}$ to be a Delaunay triangulation.)   □

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.

Keep, Jump, & Walk. Classical Jump & Walk uses a simple sequential data structure to store the random sample of $\mathcal{T}$ and assumes $\mathcal{F}(n) = O(n^{1/d})$. It is possible to 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. With a list, keep step takes $K(k) = O(1)$ and jump step takes $J(k) = O(k)$. Then, using Lemma 56 and taking $k = n^{1/(d+1)}$ landmarks amongst the queries ensures an expected 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 convex hull of $\mathcal{T}$; details on the Distribution Condition can be found in Section 8.1).

Outside this classical framework, Jump & Walk has some interests, even with weaker hypotheses; i.e., without a so good distribution of vertices. In general, considering Lemma 56, 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 $\mathcal{F}(n/k)$ [93, 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 46 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 $O(n^{3/8})$ in 3D; this is verified experimentally in Section 8.5, and should be applied in surface reconstruction applications.

Keep, Walk, & Walk. In Keep, Walk, & Walk, the data structure to store the landmarks is a Delaunay triangulation $\mathcal{L}$, in which it is possible to walk. Assuming a random order on the landmarks, inserting or deleting a landmark after location takes $O(1)$; then keep step 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 56, $W(k) = k^{-1/d} \cdot \mathcal{F}(n) = O(k^{-1/d} \cdot n^{1/d})$, 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 Keep, 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 [93].

Storing the queries in a Delaunay hierarchy may have some interesting effects: If the region $\mathcal{C}$ of $\mathcal{T}$ has some bad behavior $\mathcal{F}(n) \gg n^{1/d}$, we can get interesting query time to the price of polynomial storage. 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 56 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.

In practice, neither Keep, Walk, & Walk nor Delaunay Hierarchy of Queries work well; the reason is the relatively big constants hidden in the Big-O notation of the point insertion complexity. Further results for these strategies are omitted. However, Keep, Jump, & Walk performs well; experiments are presented in Section 8.5.3.

8.4   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 S in the space, we assume that the expected size of the Delaunay triangulation of a random sample of size r of S has linear size. The hypothesis is always verified in 2D, and proved in several other situations: points randomly distributed in space [108] or on an hypersurface [125, 34, 35, 26]. The Delaunay hierarchy [93] constructs random samples $S = S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots \supseteq S_h$ such that $Prob(p \in S_{i+1} \mid p \in S_i) = 1/\alpha$ for some constant $\alpha > 1$. The $h+1$ Delaunay triangulations $\mathcal{DT}_i$ of S_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. More details can be found in Section 7.3.3.

If a good starting vertex $v = v_0$ in $\mathcal{DT}_0$ is known, the Delaunay hierarchy can be used in another way: From v_0 a walk starts in $\mathcal{DT}_0$ visiting cells crossed by segment $v_0 q$; the walk is stopped, either if the cell containing q is found, or if a cell having a vertex v_1 belonging to the sample S_1 is found. If the walk stops because v_1 is found, then a new walk in $\mathcal{DT}_1$ starts at v_1 along segment $v_1 q$. This process continues recursively up to the level l, where a cell of $\mathcal{DT}_l$ that contains q is found; see Figure 8.3. Finally, the hierarchy is descended as in the usual point location. Theorem 57 bounds the complexity of this procedure.
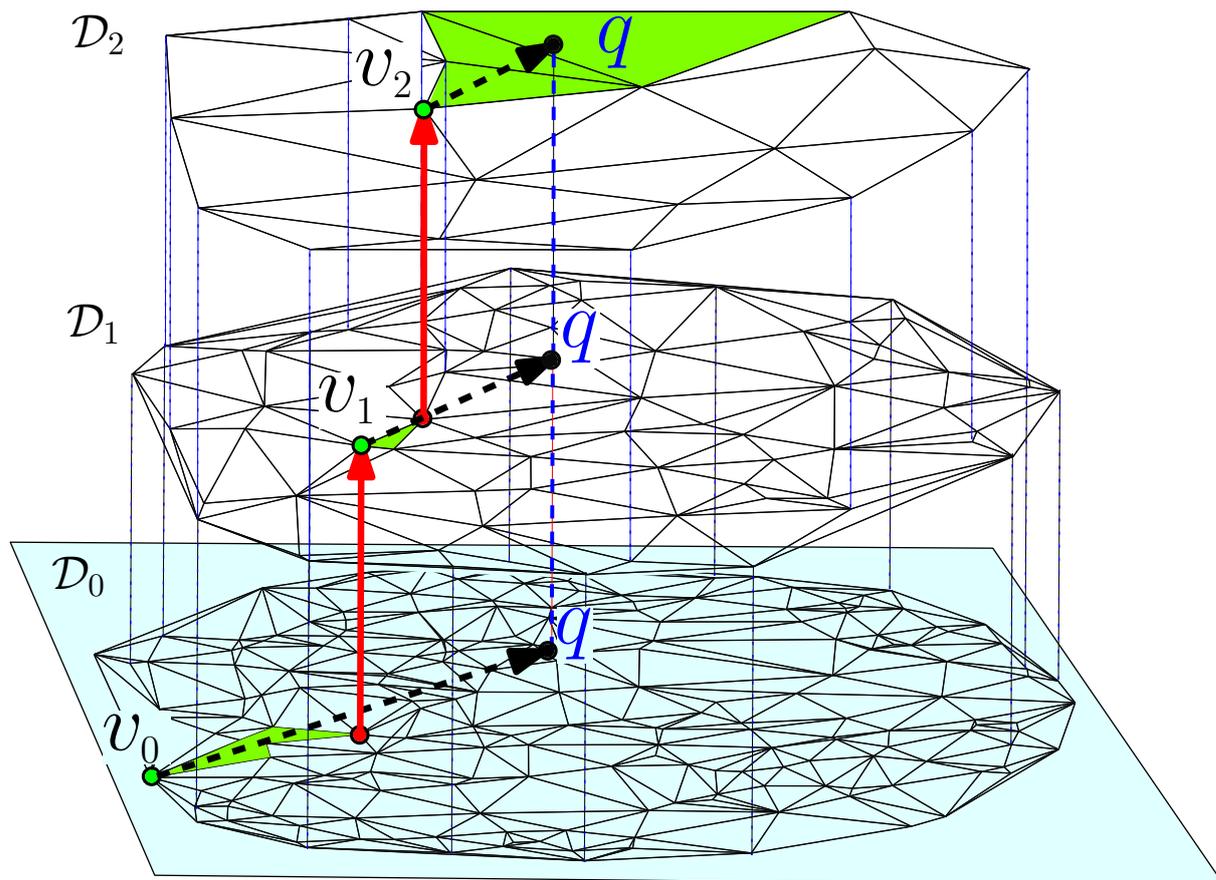
Figure 8.3: **Climbing.**

Theorem 57. *Given a set of n points S, and a convex region $\mathcal{C} \subseteq CH(S)$, such that the Delaunay triangulation of a random sample of size r of S*

(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 $\mathcal{DT}$ the Delaunay triangulation of S.

Proof. **Climbing one level.** Since the probability that any vertex of $\mathcal{DT}_i$ belongs to $\mathcal{DT}_{i+1}$ is $1/\alpha$, and that each time a new cell 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 $\mathcal{DT}_{i+1}$ is $1 + \sum_{j=0}^{\infty} j \frac{1}{\alpha} \left(1 - \frac{1}{\alpha}\right)^j = \alpha$.

Descending one level. The cost of descending one level is $O(\alpha)$ [93, Lemma 4].

Number of levels. Let w_i denote the number of edges crossed by $v_i q$ in $\mathcal{DT}_i$; the Distribution Condition gives $w_i = \mathcal{F}(n/\alpha^i) \| v_i q \| \leq \mathcal{F}(n/\alpha^i) \| v_0 q \|$. 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

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

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

Please remind that in Section 8.3, 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 $O(\log w)$. More exactly, we look at landmarks till the number of seen landmarks is smaller than the expected cost of climbing and walking from the best landmark we have seen. Using the hypotheses in Theorem 57, this cost is given by $\log w = O(\log(1 + \kappa \cdot \mathcal{F}(n) \cdot \|pq\|)) = O(\log(n\|pq\|))$; 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.

8.5   Experimental Results

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 TURTLE are taken from the Aim@shape repository [1]; the scanned models are depicted in Figure 8.4. The hardware used for the experiments described in what follows, 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 [5] 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. The walking strategy used in the section is the **stochastic walk**.

We consider the following data sets in 2D: UNIFORM SQUARE, points evenly distributed in the unit square; ANISOTROPIC SQUARE, points distributed in a square with a $\rho = x^2$ density; ELLIPSE, points evenly distributed on an ellipse, the lengths of the axes are $1/2$, and 1. We consider the following data sets in 3D: UNIFORM CUBE. Points evenly distributed in the unit cube. ANISOTROPIC CUBE. Points distributed in a cube with a $\rho = x^2$ density. ELLIPSOID. Points evenly distributed on the surface of an ellipsoid; the lengths of the ellipsoid axes are $1/3$, $2/3$, and 1. CYLINDER. Points evenly distributed on the surface of a closed cylinder. **GALAAD, POORAN'S HAND, and TURTLE.** They are data sets obtained by scanning a 3D model of a physical object.
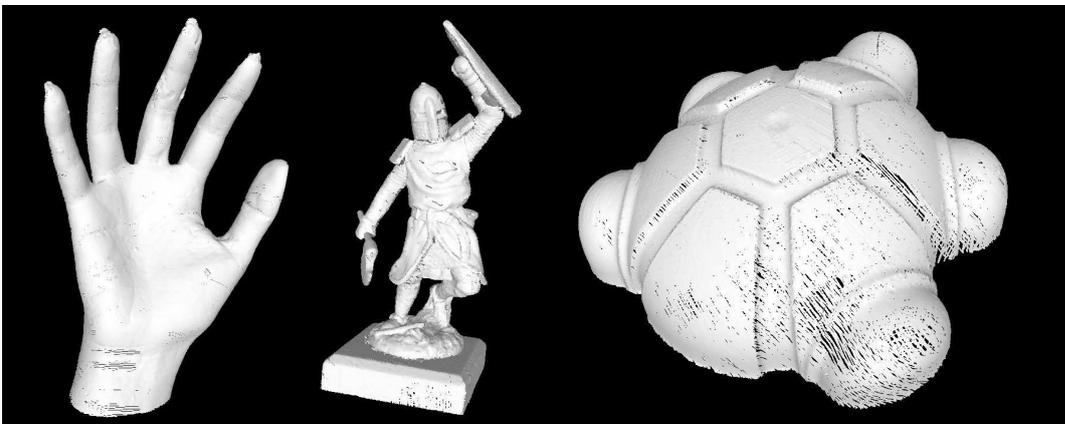

Figure 8.4: **Scanned models.**

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.

8.5.1   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 8.5 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. From this 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 8.5, 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 values of n, we draw $\mathcal{F}(n)$ in terms of the triangulation size in Figure 8.6.

If $\mathcal{F}(n)$ is clearly bounded by a polynomial on n, then curves in Figure 8.6 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 46 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 46 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.

8.5.2 k-last-points strategy

CGAL library [179] uses spatial sorting [88] to introduce a strong spatial coherence in a set of points. For each data set considered above: (i) we construct Delaunay triangulations of several sample (of 2^{20} points) of the data set; then (ii) we locate 2^{20} queries evenly distributed inside the Delaunay triangulation of the samples with the k-last-point strategy after spatial sorting the queries; finally (iii) we output the average time taken to retrieve all 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 worthwhile. Table 8.1 shows the running times on various sets for different values of k, taking $k = 4$ always improves on $k = 1$ and in some cases by a substantial amount.

8.5.3   Keep, Jump, & Walk and Keep, Jump, & Climb

In this section, we compare the performance of various point location procedures: classical Jump & Walk (**J&W**); walk starting at the previous query (**last-point**); Keep, Jump, & Walk described in Section 8.3 (**K&J&W**); descending the Delaunay hierarchy [93] (**Delaunay hierarchy**); climbing and descending the Delaunay Hierarchy (**Climb**); and
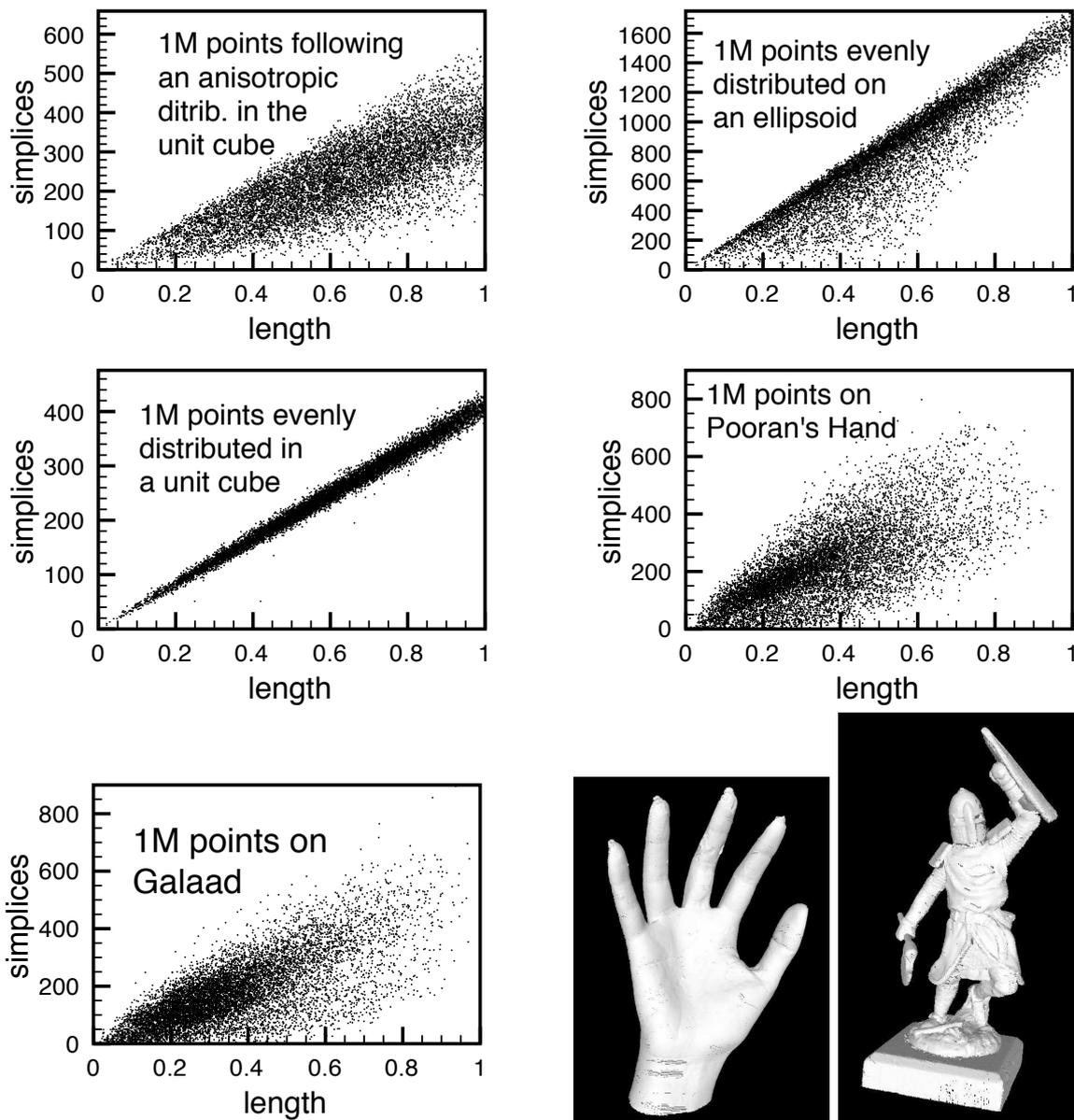
Figure 8.5: **Distribution Condition.** ♯ *of crossed tetrahedra in terms of the length of the walk. The number of points sampled in each model here is 2^{20}.*
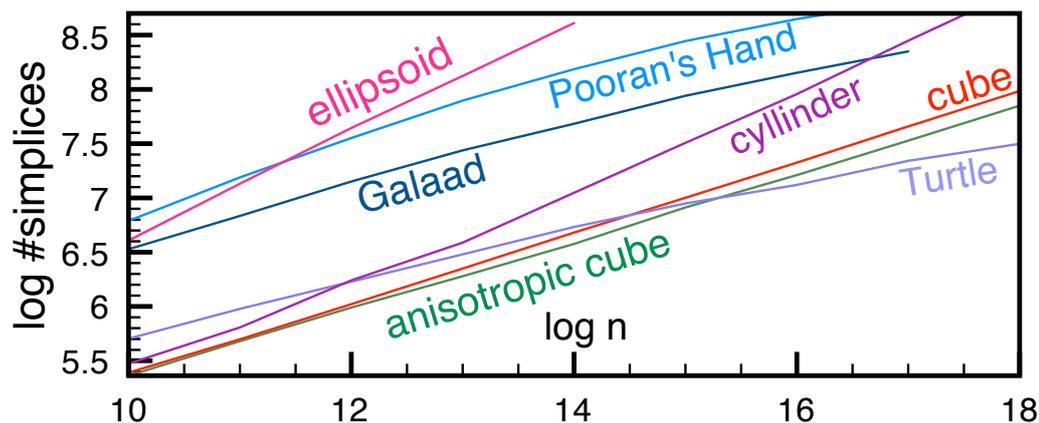
Figure 8.6: **Distribution Condition.** *Assuming $\mathcal{F}(n) = n^\alpha$, α is given by the slope of above "lines" (log here is $\log_2$).*

k	1	2	3	4	5	6	$k = 4$ improves
2D							
UNIFORM SQUARE	1.70s	1.65s	1.65s	1.65s	1.66s	1.67s	2%
ANISOTROPIC SQUARE	1.64s	1.61s	1.60s	1.60s	1.61s	1.62s	1%
ELLIPSE	**3.07s**	**2.73s**	**2.62s**	**2.56s**	**2.54s**	**2.52s**	**17%**
3D							
UNIFORM CUBE	3.57s	3.45s	3.41s	3.39s	3.40s	3.46s	5%
ANISOTROPIC CUBE	3.45s	3.35s	3.32s	3.31s	3.32s	3.39s	4%
ELLIPSOID	**6.34s**	**5.71s**	**5.48s**	**5.38s**	**5.34s**	**5.44s**	**15%**
CYLINDER	**5.43s**	**5.12s**	**5.06s**	**4.99s**	**5.03s**	**5.01s**	**8%**
POORAN'S HAND	**3.81s**	**3.63s**	**3.58s**	**3.57s**	**3.56s**	**3.63s**	**6%**
GALAAD	4.19s	4.08s	4.04s	4.03s	4.07s	4.12s	3%
TURTLE	2.15s	2.09s	2.07s	2.06s	2.05s	2.05s	5%

Table 8.1: **Static point location with space-filling heuristic plus last-k-points strategy.** *Times are in seconds.*

Keep, Jump, & Climb (**K&J&C**).[2] For the later, we use a number of $\log n$ landmarks.[3] 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 a scanned model with 2^{20} vertices inside the unit cube, we first define S_i, for $i = 0, \ldots, 20$, the 2^i vertices of $\mathcal{M}$ closest to the cube center. 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} times a random point from S_i (repetitions are allowed) and slightly perturbing these points. The perturbation actually removes duplicates and ensures that most of the queries are strictly inside a Delaunay tetrahedron and thus preventing filter failures. Figure 8.7 shows the computation times for point location and different strategies in function of i.

---

[2] The behavior of these algorithms can be experimented in a javascript demo [85].

[3] The technique described in Section 8.4 to optimize the number of landmarks is not justified when the maximal number of landmarks: $\log n$ is small enough. For our models of 2^{20} points we explore systematically 20 landmarks.

Scenario II — This scenario is designed to show how the spatial coherence of queries relates with the point location algorithms performance. Consider 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. In Figure 8.8, we compute the time to complete all 2^{20} queries generated by 1 to 20 random walkers, for different strategies. One single walker means a very strong spatial coherence; on the other hand, several walkers mean a weaker spatial coherence.

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 [223, 179].
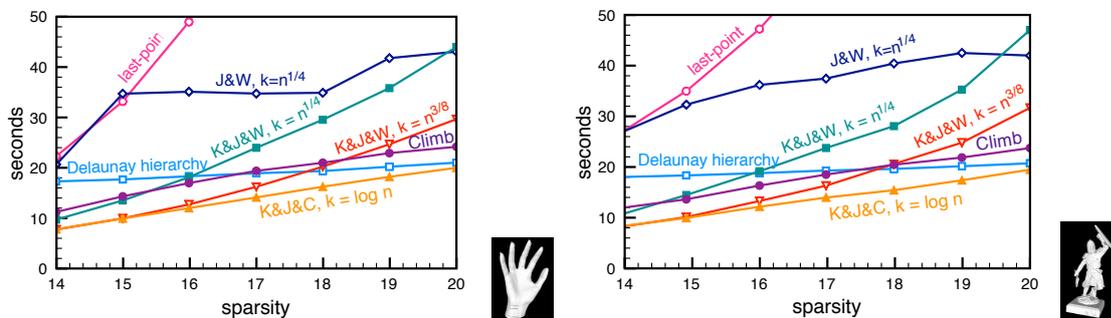
Figure 8.7: **Results for scenario I (proximity of queries).** *Computation times of the various algorithms in function of the number of points on the model enclosed by a ball centered at $(0.5, 0.5, 0.5)$ for: (a) Pooran's Hand model; and (b) Galaad model.*

From Scenario I (Figure 8.7). One can observe that Keep, Jump, & Walk actually benefits from the proximity of the queries and is clearly better than Jump & Walk and even better than the Delaunay hierarchy if the portion of $\mathcal{M}$ where the queries lie in is below 6% of the total surface of $\mathcal{M}$. Taking $n^{3/8}$ landmarks instead of $n^{1/4}$ performs clearly better which is another experimental validation of Conjecture 46 (as announced in Section 8.3). Not surprisingly, climbing and descending the hierarchy is slower than just descending the hierarchy when queries are not closer one each other, and improves with proximity. Finally, Keep, Jump, & Climb combines all the advantages and appears as the best solution in practice for this experiment.

From Scenario II (Figure 8.8). 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 case. This is indeed what happens, but Keep, Jump, & Walk and Keep, Jump, & Climb remain quite close. And, of course, when the number of walkers increases, performance of last-point strategy fall down, while Keep, Jump, & Walk/Climb still have good running times. Observe that the Keep, Jump, & Walk with $n^{3/8}$ landmarks does not seem to be strongly dependent on the number of walkers.
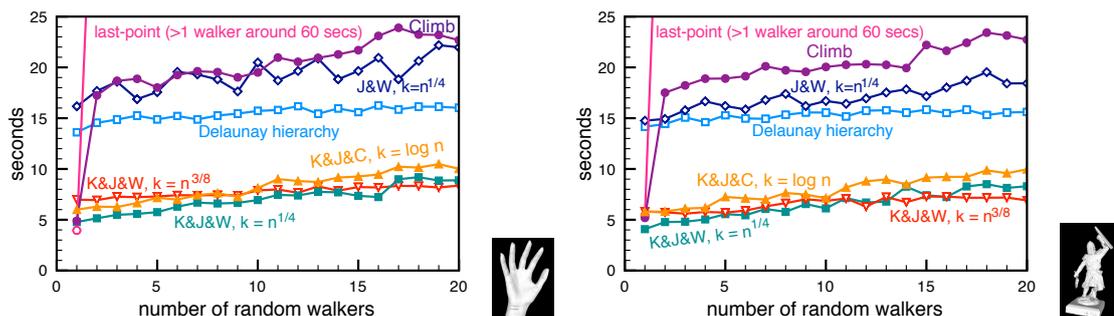
Figure 8.8: **Results for scenario II (coherence of queries).** *Computation times of the various algorithms in function of the number of parallel random walkers for: (a)* POORAN'S HAND *model; and (b)* GALAAD *model. Less random walkers mean strong spatial coherence (a single walker pushes that to an extreme).*

8.5.4   A Last Optimization: Structural Filtering

The exactness of the walk is certified by exact computations, which are classically accelerated by arithmetic filtering. To speed-up the walking procedure even more, we can use a filtering scheme called *structural filtering*, proposed by Funke et al. [120], which works at a higher level than the classical arithmetic filtering scheme. It is based on the *relaxation* of the exactness of the predicates; see Figure 8.9.

More precisely, the correctness of the visibility walk relies on the exactness of the orientation predicates; the filtering mechanism accelerates their exact computation (see Sections 2.1.2 and 2.2.4 for a review), but it remains slower than a direct evaluation with floating-point numbers (without any certification of the sign). Structural filtering extends this classical filtering mechanism to a higher level. Instead of certifying exactness at each orientation predicate, the filtering mechanism 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 improves 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 cell, and then returns that cell; or visits a number of cells equals to some given threshold, and then returns the current cell being visited.[4] This threshold mechanism avoids a possible non-termination of the relaxation phase due to rounding errors. At the end, this phase returns a cell (hopefully) not too far from the solution; see Figure 8.9.

• The second phase consists in running the visibility walk algorithm with a certified orientation predicate starting at the cell returned by the relaxation phase, until the solution is found; we call this phase the *certification phase*; see Figure 8.9.

Point location strategies described in this work are compatible with such optimization, and we could obtain around 20% speed-up for POORAN'S HAND and GALAAD models in our previous experiments; details are shown in Figures 8.10 and 8.11.

---

[4]Simplices that are not a cell, in general, can't be caught in this phase: Inexact computations are specially not reliable in degenerate cases.
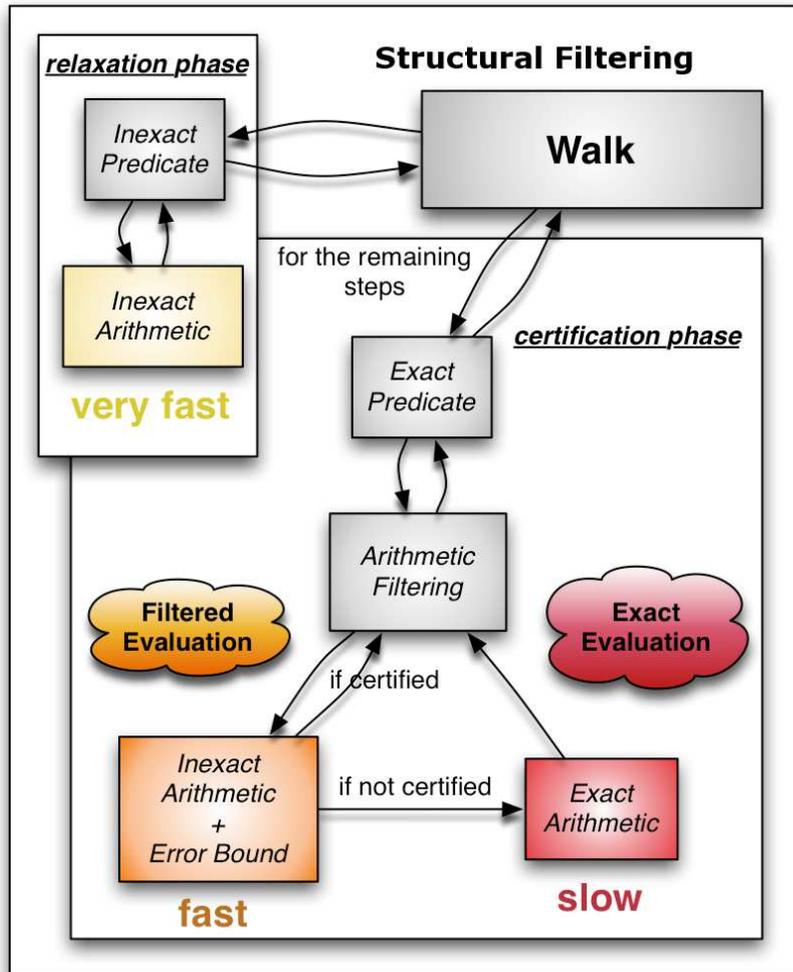
Figure 8.9: **Structural Filtering.** *The certification phase of the structural filtering scheme, is the classical arithmetic filtering scheme. Now, most of the steps are made with inexact arithmetic (very fast). One may contrast this figure with Figure 2.1, which describes the usual arithmetic filtering scheme.*

8.6   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 can handle 1M queries in a 1M points triangulation in about 15 seconds for various scenarios. We proposed Keep, Jump, & Climb: It uses the Delaunay hierarchy in a different way, which is never slower and often 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 8.8, the running times vary from around 5s to around 10s depending on the query coherence;
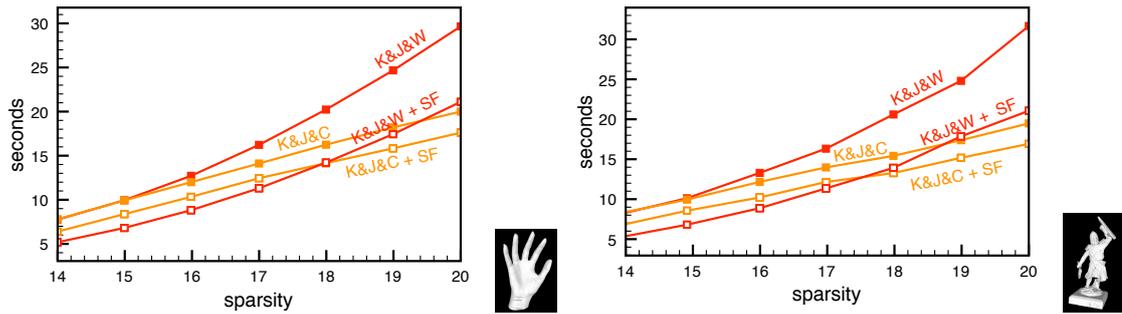
Figure 8.10: **Scenario I: with and without structural filtering (SF).**
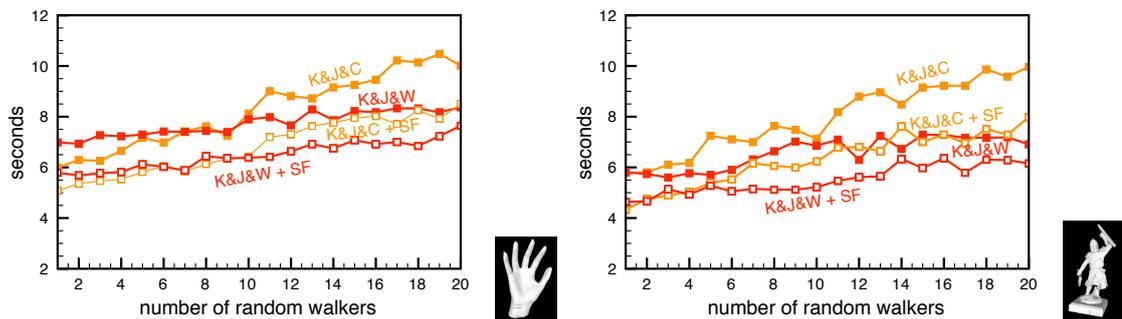
Figure 8.11: **Scenario II: with and without structural filtering (SF).**

this should be contrasted with the classical Delaunay hierarchy, which achieves around 15s.) By using structural filtering, we improve our performances by an additional 20%; structural filtering is a filtering mechanism, which allows to ensure robustness to numerical issues in a cheaper way than usual arithmetic filtering of the predicates.

One of our main tool in the theoretical analysis of our strategies 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 $n^{1/4}$. 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 [143, 91], but requires some reasonable hypotheses on the point set.

As a final remark, we insist on the dichotomy between the straight and stochastic walk. The straight walk is used in theoretical analysis for the simplicity and the available previous results, while the visibility walk is used in practice, since it is faster and easier to implement. Thus an interesting research direction is deriving stronger theoretical basis for the stochastic walk.

8.7   Open Problems

The Distribution Condition brings several questions for the computational geometers. The first question is the one raised by Conjecture 46:

Problem 58. *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 46 has an affirmative answer, then walking on such triangulations does not depend on the ambient dimension, but only on the manifold dimension.

Figure 8.5 and 8.6 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.

Problem 59. *What hypotheses a sampling of a bounded δ-dimensional manifold embedded in the d-dimensional space should verify such that the Delaunay triangulation satisfy the Distribution Condition with $\mathcal{F}(n) = n^{1/\delta}$?*

Recently Connor and Kumar [79] has been able to produce a practical point location algorithm in the plane and a k-nearest neighbor graph construction algorithm [78]. Their work relies in a well-known hypothesis called the *constant factor expansion* hypothesis. Let S be a finite set of n points in $\mathbb{R}^d$, $\mathcal{B}(c, r)$ be the ball with radius r centered at c, and $\mathcal{NN}_k(q)$ the k-th nearest neighbor of q. Then the *constant factor expansion* hypothesis requires that, for any point q lying in some region $\mathcal{C}$ and any $k = 1, \ldots, n$, the number of points of S enclosed by $\mathcal{B}(q, 2 \cdot \|q\mathcal{NN}_k(q)\|) \leq \gamma k$, where $\gamma = O(1)$. The constant factor expansion hypothesis and the Distribution Condition seem to be related, and a positive answer to the following question may enable the sharing of results derived from one or another.

Problem 60. *Is the Distribution Condition related, in some sense, to the constant factor expansion hypothesis?*

In the plane, by climbing and descending the Delaunay hierarchy, one can obtain a $o(\log n)$ distribution-sensitive point location algorithm (in expectation and amortization), as long as the triangulation scheme and distribution of points satisfy the Distribution Condition with $\mathcal{F} = o(n^\epsilon), \forall \epsilon > 0$, in the domain the queries lie in. Such triangulation schemes with sub-polynomial Distribution Condition seems quite restrictive but they indeed exist as shown by the example depicted in Figure 8.12.

Problem 61. *What is the least restrictive set of hypotheses on the triangulation and on the queries, such that a $o(\log n)$ distribution-sensitive point location algorithm is possible?*

Problem 62. *In the plane, is it possible to climb in the Delaunay hierarchy with a good complexity and with less restrictive hypotheses than in Theorem 57?*

Problem 63. *Let Δ be any triangulation scheme (such as Delaunay, Regular, Constrained, …), let ρ be any distribution of points with compact support $\Sigma \subset \mathbb{R}^d$, and let $\mathcal{C}$ be any region inside Σ with positive volume. For a triangulation $\mathcal{T}$ of n points following distribution ρ and built upon the triangulation scheme Δ, does the Distribution Condition necessarily hold almost everywhere in $\mathcal{C}$, for some polynomial $\mathcal{F}(n)$ (say $\mathcal{F}(n) = O\left(n^{\lceil d/2 \rceil}\right)$) ?*

Note that regions that does not satisfy the Distribution Condition exist; see Figure 8.12 (the circle). However, the region has no volume. We could not find an example of triangulation scheme and distribution of points, such that the Distribution Condition does not hold for some polynomial $\mathcal{F}(n)$, and a region with positive volume.

Finally, we insist once again in the dichotomy between straight walk and stochastic walk.

Problem 64. *What is the actual complexity of the stochastic walk?*
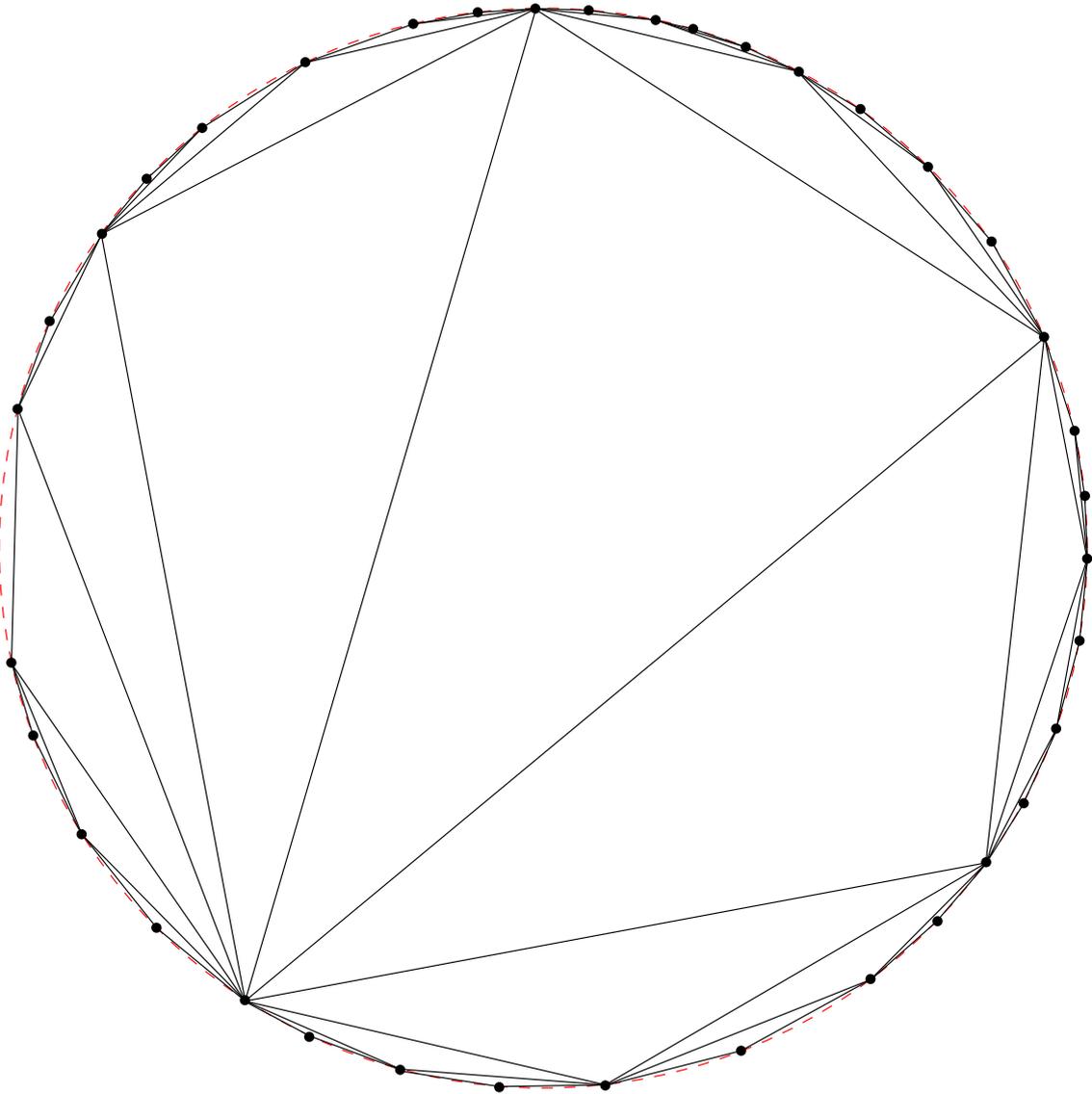
Figure 8.12: **Fractal-like scheme.** *The triangulation scheme above, for points uniformly distributed on the circle, satisfy the Distribution Condition with $\mathcal{F} = O(\log n)$ for any closed region inside the circle. However, if we take a segment s intersecting the circle, then as $n \to \infty$, s intersects the same number of cells regardless of its size, violating the Distribution Condition.*

Part IV

Conclusion

Chapter 9

A Few Last Words

"Rien ne se perd, rien ne se crée, tout se transforme." — *Antoine-Laurent de Lavoisier.*[1]

9.1 Summary

In this thesis, we proposed several new practical ways to speed-up some of the most important operations in a Delaunay triangulation.[2] We concentrated ourselves on the following aspects:

- **Efficiency.** Solutions presented in this work have some added value; they are faster than at least one important state-of-the-art solution.

- **Robustness.** Solutions presented in this work solve the problem they aim to solve, for any valid input.

- **Simplicity.** Solutions presented in this work are simple to implement.

The simplest solutions we designed were often the ones with the best performance. The following sections summarize our work.

---

[1] Nothing is lost, nothing is created, everything is transformed.

[2] We also obtained new theoretical results on the growth rate of some greedy spanning tree embedded in $\mathbb{R}^d$ (see Chapter 6). However, this is a side-result, where the first purpose, in the context of this thesis, is to provide enough background for Chapter 8.

9.1.1   Delaunay Triangulations of Points On or Close to a Sphere

Idea: *Plug slightly modified predicates into a slightly modified regular triangulation to:*
• build the Delaunay triangulation of the rays emanating from the center of the sphere to the input points;
• build the convex hull of the input points.

In Chapter 3, we proposed two approaches for computing the Delaunay triangulation of points on a sphere, or of rounded points close to a sphere, both based on the classic incremental algorithm initially designed for the plane. We implemented the two approaches in a fully robust way, building upon existing generic algorithms provided by the CGAL library. The efficiency and scalability of the method was shown by benchmarks. The second approach provides a two factor speed-up on the state-of-the-art approaches.

Chapter 3 brought the following important news: (i) when most of the input points lie on their convex hull, then a two-dimensional structure is more adequate than a three-dimensional one (for the incremental construction); (ii) otherwise, then a three-dimensional is more adequate.

9.1.2   Filtering Relocations on Delaunay Triangulations

Idea: *Compute a tolerance for each point representing how far it can freely move without affecting the "Delaunayness" of the triangulation. Then when relocating points, before any combinatorial change, check the tolerance in order to see whether it is necessary to update the combinatorial structure of the triangulation or not.*

Updating a Delaunay triangulation when its vertices move is a bottleneck in several domains of application. Rebuilding the whole triangulation from scratch is surprisingly a viable option compared to relocating the vertices. However, when all points move with a small magnitude, or when only a fraction of the vertices moves, rebuilding is no longer the best option. Chapter 5 considers the problem of efficiently updating a Delaunay triangulation when its vertices are moving under small perturbations. The main contribution is a filtering scheme based upon the concept of vertex tolerances.

We conducted several experiments to showcase the behavior of the algorithm for a variety of data sets. The experiments showed that the algorithm is particularly relevant when the magnitude of the displacement keeps decreasing while the tolerances keep increasing. Such configurations occur in convergent schemes such as the Lloyd iterations. For the latter, and in two dimensions, the algorithm presented performs up to an order of magnitude faster than rebuilding.

In three dimensions, and although rebuilding the whole triangulation at each time stamp can be as fast as our algorithm when all vertices move, our solution is fully dynamic and outperforms previous dynamic solutions. Such a dynamic property is required for some applications, e.g practical variational mesh generation and optimization techniques. The result in Chapter 5 makes it possible to go further on the number of iterations so as to produce higher quality meshes.

9.1.3 Distribution-Sensitive Point Location

Idea: *For a batch of query points, the main idea is to use previous queries to improve the location of the current one. Additionally, instead of descending the Delaunay hierarchy from the highest level, in the classical way, one could "climb" the hierarchy and then descend the hierarchy at a possibly lower level. The combination of these two ideas leads to a novel point location algorithm called Keep, Jump, & Climb, which appears to take the best of these ideas both in theory and in practice. Analyses are possible by requiring the Distribution Condition to hold when relevant.*

Point location in spatial subdivision is one of the most studied problems in computational geometry. In the case of triangulations of $\mathbb{R}^d$, we revisited the problem to exploit a possible coherence between the query points.

For a single query, walking in the triangulation is a classical strategy with good practical behavior and expected complexity $O(n^{1/d})$ if the points are evenly distributed. Based upon this strategy, in Chapter 8, we analyzed, implemented, and evaluated 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 location of the current one. In practice, Keep, Jump, & Walk seems to be a very competitive method to locate points in a triangulation.

Regarding point location in a Delaunay triangulation, in Chapter 8, we showed how the Delaunay hierarchy can be used to answer, under some hypotheses, a query q with a $O(\log \sharp(pq))$ randomized expected complexity, where p is a previously located query and $\sharp(s)$ indicates the number of simplices crossed by the line segment s.

We combined 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—in our experiments, it is faster than the Delaunay hierarchy regardless of the spatial coherence of queries, and significantly faster (more than twice speed-up factor) when queries have reasonable spatial coherence.

And last but not least, we introduced the Distribution Condition in Chapter 8, which allowed us to analyze the algorithms above. Roughly speaking, 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. This Distribution Condition seems to be realistic; some indicatives are shown in Chapter 8.

9.2 Perspectives

- **Parallelism.** While parallelism did not take its place in this thesis, I believe in its future. More precisely, I think that improvements optimizing the parallelization aspects of an algorithm is more relevant to improve *speed* than fine-tuning constants. Clock rates do not grow at the same pace as ten years ago. Instead, it seems possible to increase the number of different processors working together to solve one problem. Sometimes when memory are not necessarily shared between the

processors, surprisingly the speed-up can be even proportionally higher than the number of processors; this happens mainly because of the reduction of cache failures.

I believe however, that algorithm complexity remains the main source of speed-up. The n term (the size of the input) will probably keep to increase with time. For instance, when successive queries have no spatial coherence, Keep, Jump, & Walk for $n < 30,000$ is often faster than Keep, Jump, & Climb, however at $n = 10^6$ the situation changes. Interesting meshes nowadays has between one and three orders of magnitude more vertices than interesting meshes of twenty years ago.

- **Simplicity.** I believe that simplicity should be sought. During my thesis, the simplest algorithms were often the best ones in practice; such a recurring fact should not be simply demoted as coincidence.

  In Chapter 5, we designed a filtering scheme for relocation in a Delaunay triangulation based on the tolerance of a vertex. We designed several other more involved tolerance regions, and filtering schemes; any of them being half as fast as the scheme described in Chapter 5.

  In Chapter 8, we proposed several distribution-sensitive point location algorithms based on Keep, Jump, & Walk; however, Keep, Jump, & Walk (and the related algorithms) was not the only strategy we sought to allow distribution-sensitiveness. We also tried e.g., to include cells visited during a previous walk, and optimizing some locality criteria, as landmarks. The winner was once again the simplest: Keep, Jump, & Walk.

- **Filtering.** All the works in this thesis benefit somehow from filtering. Approaches in Chapter 3 benefit from static, semi-static and dynamic arithmetic filtering. The method designed in Chapter 5 is a new filtering scheme itself. Chapter 8 benefit from static, semi-static, dynamic, and structural filtering.

- **Distribution Condition.** The Distribution Condition allows one to relate the complexity of a walk in the triangulation with the size of a segment. When strategies can be expressed as several walks, than the complexity of the strategy relates with the growth rate of the corresponding graph in the space; this is a very important shortcut. The Distribution Condition seems to be a realistic condition (if not a necessary condition), and it can be useful for: (i) further analyses of distribution-sensitive algorithms, and (ii) a better understanding of the properties of a triangulation in $\mathbb{R}^d$. Perhaps the function $\mathcal{F}$ in the Distribution Condition is a nice descriptor of the complexity of a given distribution of points in $\mathbb{R}^d$ for a triangulation scheme.

9.3 Open Problems

To conclude this work, I briefly list here some of the open problems stated along the run I find particularly interesting.

- Problem 19 (Chapter 5): *Is there in three dimensions a static filtering algorithm which performs an order of magnitude faster than rebuilding in practice?*

- Problem 21 (Chapter 5): *Is the Vertex Tolerance Filtering algorithm an instance of a more general time stamp point relocation framework?*

- Problem 58 (Chapter 8): *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?*

- Problem 63 (Chapter 8): *Let Δ be any triangulation scheme (such as Delaunay, Regular, Constrained, ...), let ρ be any distribution of points with compact support $\Sigma \subset \mathbb{R}^d$, and let $\mathcal{C}$ be any region inside Σ with positive volume. For a triangulation $\mathcal{T}$ of n points following distribution ρ and built upon the triangulation scheme Δ, does the Distribution Condition necessarily hold almost everywhere in $\mathcal{C}$, for some polynomial $\mathcal{F}(n)$ (say $\mathcal{F}(n) = O\left(n^{\lceil d/2 \rceil}\right)$)?*

- Problem 64 (Chapter 8): *What is the actual complexity of the stochastic walk?*

Bibliography

[1] AIM@SHAPE Shape Repository v4.0/
http://shapes.aimatshape.net.

[2] Geometric software: Three-dimensional convex hulls.
http://home.mims.meiji.ac.jp/∼sugihara/opensoft/opensofte.html.

[3] Hull, a program for convex hulls.
http://www.netlib.org/voronoi/hull.html.

[4] Qhull.
http://www.qhull.org/.

[5] Cgal, Computational Geometry Algorithms Library.
http://www.cgal.org.

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

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

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

[9] IEEE standard for floating-point arithmetic. *IEEE standard 754-2008*, 1–58, 2008.

[10] M. A. Abam, M. de Berg, S.-H. Poon, and B. Speckmann. Kinetic collision detection for convex fat objects. In Proceedings of the 14th Annual European Symposium on Algorithms, 4–15, 2006.

[11] M. A. Abam, P. K. Agarwal, M. de Berg, and H. Yu. Out-of-order event processing in kinetic data structures. In Proceedings of the 14th Annual European Symposium on Algorithms, 624–635, 2006.

[12] U. A. Acar, G. E. Blelloch, K. Tangwongsan, and D. Türkoğlu. Robust kinetic convex hulls in 3D. In Proceedings of the 16th Annual European Symposium on Algorithms, 29–40, 2008.

[13] U. A. Acar, G. E. Blelloch, K. Tangwongsan, and J. L. Vittes. Kinetic algorithms via self-adjusting computation. In Proceedings of the 14th Annual European Symposium on Algorithms, 636–647, 2006.

[14] U. Adamy and R. Seidel. On the exact worst case query complexity of planar point location. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 609–618, 1998.

[15] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang. International Journal of Robotics Research, 83–96, 2000.

[16] P. K. Agarwal, J. Basch, M. de Berg, L. J. Guibas, and J. Hershberger. Lower bounds for kinetic planar subdivisions. In *Proceedings of the 15th ACM Annual Symposium on Computational Geometry*, 247–254, 1999.

[17] P. K. Agarwal, J. Gao., L. J. Guibas, H. Kaplan, V. Koltun, N. Rubin and M. Sharir. Kinetic stable Delaunay graphs, In *Proceedings of the 26th ACM Annual Symposium on Computational Geometry*, 127–136, 2010.

[18] P. K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, 1–47. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[19] P. K. Agarwal, Y. Wang, and H. Yu. A 2D kinetic triangulation with near-quadratic topological changes. In *Proceedings of the 20th ACM Annual Symposium on Computational Geometry*, 180–189, 2004

[20] G. Albers, L. J. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *International Journal of Computational Geometry and Applications*, 8:365–380, 1998.

[21] D. Aldous and J. M. Steele. Asymptotics for euclidean minimal spanning trees on random points. *Probability Theory and Related Fields*, 92:247–258, 1992.

[22] G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Computational Geometry: Theory and Applications*, 36(2):144–158, 2007.

[23] P. Alliez, E Colin de Verdière, O. Devillers, M. Isenburg. Isotropic surface remeshing. In *Shape Modeling International*, 49–58, 2003.

[24] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24:617–625, 2005.

[25] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *Proceedings of the 8th ACM-SIAM Symposium Discrete Algorithms*, 528–537, 1997.

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

[27] N. Amenta, Sunghee Choi, and G. Rote. Incremental constructions con brio. In *Proceedings of the 19th Annual Symposium on Computational Geometry*, 211–219, 2003.

[28] C. R. Aragon and R. Seidel. Randomized search trees. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, 540–545, 1989.

[29] D. B. Arnold and W. J. Milne. The use of Voronoi tessellations in processing soil survey results. *IEEE Computer Graphics and Applications*, 4:22–28, 1984.

[30] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the 18th ACM-SIAM Symposium Discrete Algorithms*, 1027–1035, 2007.

[31] S. Arya, T. Malamatos, and D. M. Mount. Entropy-preserving cuttings and space-efficient planar point location. In *Proceedings of the 12th ACM-SIAM Symposium Discrete Algorithms*, 256–261, 2001.

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

[33] S. Arya, T. Malamatos, and D. M. Mount. Optimal expected-case planar point location. *SIAM Journal on Computing*, 37(2):584–610, 2008.

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

[35] D. Attali, J.-D. Boissonnat, and A. Lieutier. Complexity of the Delaunay triangulation of points on surfaces: The smooth case. In *Proceedings of the 19th ACM Annual Symposium on Computational Geometry*, 237–246, 2003.

[36] F. Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing*, 16:78–96, 1987.

[37] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[38] D. H. Bailey, J. M. Borwein, and R. E. Crandall. Box integrals. *Journal of Computational and Applied Mathematics*, 206(1):196–208, 2007.

[39] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. *Computational Geometry: Theory and Applications*, 27(3):211–235, 2004.

[40] J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proceedings of the 13th ACM Annual Symposium on Computational Geometry*, 388–390, 1997.

[41] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Journal of Algorithms*, 747–756, 1998.

[42] V. H. F. Batista, D. L. Millman, S. Pion, and J. Singler. Parallel geometric algorithms for multi-core computers. In *Proceedings of the 25th ACM Annual Symposium on Computational Geometry*, 217–226, 2009.

[43] J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55:299–327, 1959.

[44] M. Berger. The space of spheres. In *Geometry (vols. 1-2)*, 349–361, 1987. Springer. Translated by M. Cole, and S. Levy.

[45] M. Bern, D. Eppstein, and F. Yao. The expected extremes in a Delaunay triangulation. *International Journal of Computational Geometry and Applications*, 1:79–91, 1991.

[46] D. K. Blandford, G. E. Blelloch, and C. Kadow. Engineering a compact parallel Delaunay algorithm in 3D. In *Proceedings of the 22th ACM Annual Symposium on Computational Geometry*, 292–300, 2006.

[47] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112:339–354, 1993.

[48] J.-D. Boissonnat, F. Cazals, F. Da, O. Devillers, S. Pion, François Rebufat, M. Teillaud, and M. Yvinec. Programming with CGAL: The example of triangulations. In *Proceedings of the 15th ACM Annual Symposium on Computational Geometry*, 421–423, 1999.

[49] J.-D. Boissonnat, O. Devillers, and Samuel Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *Proceedings of the 25th ACM Annual Symposium on Computational Geometry*, 208–216, 2009.

[50] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete and Computational Geometry*, 8:51–71, 1992.

[51] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by H. Brönnimann.

[52] P. Bose and L. Devroye. On the stabbing number of a random Delaunay triangulation. *Computational Geometry: Theory and Applications*, 36(2):89–105, 2007.

[53] P. Bose and L. Devroye. Intersections with random geometric objects. *Computational Geometry: Theory and Applications*, 10:139–154, 1998.

[54] P. Bose and L. Devroye. Flips in planar graphs. *Computational Geometry: Theory and Applications*, 42:60–80, 2009.

[55] A. Bowyer. Computing Dirichlet tesselations. *The Computer Journal*, 24:162–166, 1981.

[56] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109:25–47, 2001.

[57] K. Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, 1979.

[58] K. Q. Brown. *Geometric transforms for fast geometric algorithms.* Ph.D. thesis, Dept. Comp. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1980. Report CMU-CS-80-101.

[59] H. Brönnimann, A. Fabri, G.-J. Giezeman, S. Hert, M. Hoffmann, L. Kettner, S. Schirra, and S. Pion. 2D and 3D geometry kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.6 edition, 2010.

[60] K. Buchin. Constructing Delaunay triangulations along space-filling curves. In Proceedings of the 17th Annual European Symposium on Algorithms, 119–130, 2009.

[61] M. Caroli and M. Teillaud. Computing 3D periodic triangulations. In *Proceedings of the 17th Annual European Symposium on Algorithms*, 37–48, 2009.

[62] M. Caroli and M. Teillaud. 3D periodic triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual.* 3.6 edition, 2010.

[63] F. Cazals and J. Giesen. Delaunay triangulation based surface reconstruction. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces.* Springer-Verlag, Mathematics and Visualization, 2006.

[64] CGAL Editorial Board. *CGAL User and Reference Manual*, 3.6 edition, 2010. www.cgal.org.

[65] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6(5):485–524, 1991.

[66] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete and Computational Geometry*, 10:377–409, 1993.

[67] L. Chen. Mesh smoothing schemes based on optimal Delaunay triangulations. In *International Meshing Roundtable*, 109–120, 2004.

[68] L. Chen and J. Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[69] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *Journal of the Association for Computing Machinery*, 47:883–904, 2000.

[70] L. P. Chew. Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comp. Sci., Dartmouth College, Hanover, NH, 1990.

[71] L. P. Chew. Near-quadratic bounds for the L_1 Voronoi diagram of moving points. *Computational Geometry: Theory and Applications*, 7:73–80, 1997.

[72] L. P. Chew, K. Kedem, M. Sharir, B. Tagansky, and E. Welzl Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete algorithms*, 197–204, 2001.

[73] P. Cignoni, C. Montani, and R. Scopigno. A merge-first divide and conquer algorithm for E^d Delaunay triangulations. Technical report, Consiglio Nazionale delle Ricerche, Pisa, Italy, 1994.

[74] K. L. Clarkson. Safe and effective determinant evaluation. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, 387–395, 1992.

[75] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3(4):185–212, 1993.

[76] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[77] R. Cole. Searching and storing similar lists. *Journal of Algorithms*, 7:202–220, 1986.

[78] M. Connor and P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2010.

[79] M. Connor and P. Kumar. Practical nearest neighbor search in the plane. In *Proceedings of the 9th International Symposium on Experimental Algorithms*, 501–512, 2010.

[80] H. S. M. Coxeter. A geometrical background for de Sitter's world. *American Mathematical Monthly*, 50:217–228, 1943.

[81] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1063–1075, 1989.

[82] M. de Berg, M. van Kreveld, M. Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

[83] P. M. M. de Castro, F. Cazals, S. Loriot, and M. Teillaud. Design of the CGAL 3D Spherical Kernel and application to arrangements of circles on a sphere. *Computational Geometry: Theory and Applications*, 42(6-7):536–550, 2009.

[84] P. M. M. de Castro, F. Cazals, S. Loriot, and M. Teillaud. 3D spherical geometry kernel. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[85] P. M. M. de Castro and O. Devillers.
Demo: Point Location Strategies, 2010.
http://www-sop.inria.fr/geometrica/demo/point_location_strategies.

[86] H. L. de Cougny and M. S. Shephard. Surface meshing using vertex insertion. In *Proceedings of the 5th International Meshing Roundtable*, 243–256, 1996.

[87] J.-B. Debard, R. Balp, and R. Chaine. Dynamic Delaunay tetrahedralisation of a deforming surface. *The Visual Computer*, page 12 pp, 2007.

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

[89] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

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

[91] E. D. Demaine, J. Iacono, and S. Langerman. Proximate point searching. *Computational Geometry: Theory and Applications*, 28(1):29–40, 2004.

[92] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *International Journal of Computational Geometry and Applications*, 2(1):97–111, 1992.

[93] O. Devillers. The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13:163–180, 2002.

[94] O. Devillers. On deletion in Delaunay triangulation. *International Journal of Computational Geometry and Applications*, 12:193–205, 2002.

[95] O. Devillers, G. Liotta, F. P. Preparata, and R. Tamassia. Checking the convexity of polytopes and the planarity of subdivisions. *Computational Geometry: Theory and Applications*, 11:187–208, 1998.

[96] O. Devillers, S. Meiser, and M. Teillaud. The space of spheres, a geometric tool to unify duality results on Voronoi diagrams. In *Abstracts 8th European Workshop Comp. Geom.*, 45–49. Utrecht University, 1992.

[97] O. Devillers, S. Pion, and M. Teillaud. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13:181–199, 2002.

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

[99] O. Devillers and M. Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *Proceedings of the 14th ACM-SIAM Symposium Discrete Algorithms*, 313–319, 2003.

[100] L. Devroye, C. Lemaire, and J.-M. Moreau. Expected time analysis for Delaunay point location. *Computational Geometry: Theory and Applications*, 29:61–89, 2004.

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

[102] A. K. Dewdney. Complexity of nearest neighbour searching in three and higher dimensions. Report 28, Dept. Comp. Sci., Univ. Western Ontario, London, ON, 1977.

[103] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5:181–186, 1976.

[104] S. Drysdale. Voronoi Diagrams: Applications from Archaology to Zoology. http://www.ics.uci.edu/∼eppstein/gina/scot.drysdale.html

[105] Q. Du and M. Emelianenko. Acceleration schemes for computing centroidal Voronoi tessellations. *Numerical Linear Algebra with Applications*, 13, 2006.

[106] Q. Du and M. Emelianenko. Recent progress in robust and quality Delaunay mesh generation. *Journal of Computational and Applied Mathematics*, 195(1):8–23, 2006.

[107] Q. Du, M. Emelianenko, and L. Ju. Convergence of the Lloyd algorithm for computing centroidal Voronoi tessellations. *SIAM Journal on Numerical Analysis*, 44(1):102–119, 2006.

[108] R. A. Dwyer. Convex hulls of samples from spherically symmetric distributions. *Discrete Applied Mathematics*, 31(2):113–132, 1991.

[109] R. A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. In *Proceedings of the 5th ACM Annual Symposium on Computational Geometry*, 326–333, 1989.

[110] H. Edelsbrunner. Triangulations and meshes in computational geometry. *Acta Numerica*, 9:133-213, 2000.

[111] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge, 2001.

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

[113] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[114] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete and Computational Geometry*, 1:25–44, 1986.

[115] E. Fogel and O. Setter. Software for Voronoi diagram on a sphere. Personal communication.

[116] E. Fogel, O. Setter, and D. Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. In *Abstracts of 24th European Workshop on Computational Geometry*, 83–86, 2008.

[117] E. Fogel and M. Teillaud. Generic programming and the CGAL library. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006.

[118] J. F. Fohlmeister. Plate-kinematic constraints from mantle bénard convection. *Pure and Applied Geophysics*, 161:723–753, 2004.

[119] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[120] S. Funke, K. Mehlhorn, and S. Näher. Structural filtering: A paradigm for efficient and exact geometric programs. *Computational Geometry: Theory and Applications*, 31(3):179–194, 2005.

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

[122] J. Garcia-Lopez and P. A. Ramos. Fitting a set of points by a circle. In *Proceedings of the 13th ACM Annual Symposium on Computational Geometry*, 139–146, 1997.

[123] M. R. Garey, D. S. J.son, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175–179, 1978.

[124] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[125] M. J. Golin and H.-S. Na. On the average complexity of 3D-Voronoi diagrams of random points on convex polytopes. *Computational Geometry: Theory and Applications*, 25:197–231, 2003.

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

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

[128] L. J. Guibas and D. Russel. An empirical comparison of techniques for updating Delaunay triangulations. In *Proceedings of the 20th ACM Annual Symposium on Computational Geometry*, 170–179, 2004.

[129] L. J. Guibas. Kinetic data structures: a state of the art report. In *Workshop on the Algorithmic Foundations of Robotics*, 191–209, 1998.

[130] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[131] L. J. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points in the plane. In *Proceedings of the 17th International Graph-Theoretic Concepts in Computer Science*, 570:113–125, 1992.

[132] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[133] L. J. Guibas, F. Xie, and L. Zhang. Kinetic collision detection: Algorithms and experiments. In *IEEE International Conference on Robotics and Automation*, 2903–2910, 2001.

[134] I. Haran and D. Halperin. An experimental study of point location in planar arrangements in CGAL. *ACM Journal on Experimental Algorithmics*, 13:2.3–2.32, 2009.

[135] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, 313–319, 1984.

[136] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[137] M. Hemmer, S. Hert, L. Kettner, S. Pion, and S. Schirra. Number types. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[138] S. Hert and S. Schirra. 3D convex hulls. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[139] S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17:157–184, 1982.

[140] J. Iacono. Improved upper bounds for pairing heaps. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, 32–45, 2000.

[141] J. Iacono. Optimal planar point location. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete algorithms*, 340–341, 2001.

[142] J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, 29(1):19–22, 2004.

[143] J. Iacono and S. Langerman. Proximate planar point location. In *Proceedings of the 19th ACM Annual Symposium on Computational Geometry*, 220–226, 2003.

[144] B. Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(4):718–741, 1989.

[145] H. Kaplan, N. Rubin, and M. Sharir. A kinetic triangulation scheme for moving points in the plane. In *Proceedings of the 26th ACM Annual Symposium on Computational Geometry*, 137–146, 2010.

[146] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*, 61–70, 2006.

[147] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry: Theory and Applications*, 40:61–78, 2008.

[148] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[149] D. G. Kirkpatrick, J. Snoeyink, B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications*, 12:3–27, 2002.

[150] M. Klazar. On the maximum lengths of Davenport–Schinzel sequences. In *Contemporary Trends in Discrete Mathematics*, 169–178, 1999.

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

[152] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[153] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, 51–54, 1999.

[154] M. Larrea, D. Urribarri, S. Martig, and S. Castro. Spherical layout implementation using centroidal Voronoi tessellations. *Journal of Computing*, 1(1):81–86, 2009.

[155] C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Math. Software III*, 161–194. Academic Press, New York, NY, 1977.

[156] D. T. Lee and B. J. Schachter. Two algorithms for constructing Delaunay triangulations. *International Journal of Parallel Programming*, 9(3):219–242, 1980.

[157] S. Lee. Worst case asymptotics of power-weighted euclidean functionals. *Discrete Mathematics*, 256(1-2):291–300, 2002.

[158] C. Li, S. Pion, and C. Yap. Recent progress in exact geometric computation. *Journal of Logic and Algebraic Programming*, 64(1):85–111, 2005.

[159] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, 162–170, 1977.

[160] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. In *Proceedings of the GI Conference on Theoretical Computer Science*, 1977.

[161] Y. Liu, W. Wang, B. Levy, F. Sun, D.-M. Yan, L. Lu, and C. Yang. On centroidal Voronoi tessellation—energy smoothness and fast computation. Technical Report TR-2008-18, Department of Computer Science, The University of Hong Kong, 2008.

[162] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[163] T. Malamatos. Lower bounds for expected-case planar point location. *Computational Geometry: Theory and Applications*, 39(2):91–103, 2008.

[164] A. Maus. Delaunay triangulation and the convex hull of n points in expected linear time. *BIT Numerical Mathematics*, 24:151–163, 1984.

[165] G. Melquiond and S. Pion. Formal certification of arithmetic filters for geometric predicates. In *Proceedings of the 17th IMACS World Congress on Scientific, Applied Mathematics and Simulation*, 2005.

[166] M. Meyer, P. Georgel, and R. T. Whitaker. Robust particle systems for curvature dependent sampling of implicit surfaces. In *Proceedings of the International Conference on Shape Modeling and Applications*, 124–133, 2005.

[167] V. Milenkovic and E. Sacks. An approximate arrangement algorithm for semi-algebraic curves. In *Proceedings of the 22nd ACM Annual Symposium on Computational Geometry*, 237–246, 2006.

[168] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proceedings of the 12th ACM Annual Symposium on Computational Geometry*, 274–283, 1996.

[169] J. M. Muller. On the definition of ulp(x). Research Report 5504, INRIA, 2005. http://hal.inria.fr/inria-00070503/.

[170] K. Mulmuley. A fast planar partition algorithm, I. *Journal of Symbolic Computation*, 10(3-4):253–280, 1990.

[171] N. C. Myers. Traits: a new and useful template technique. *C++ Report*, 1995.

[172] H. S. Na, C. N. Lee, and O. Cheong. Voronoi diagrams on the sphere. *Computational Geometry: Theory and Applications*, 23:183–194, 2002.

[173] N. M. Nam, L. H. Bac, and N. V. Nam. An extreme algorithm for network-topology construction based on constrained Delaunay triangulation. *International Conference on Knowledge Systems Engineering*, 179–184, 2009.

[174] G. Nivasch. Improved bounds and new techniques for Davenport–Schinzel sequences and their generalizations. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1–10, 2009.

[175] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. J. Wiley and Sons, Chichester, UK, 1992.

[176] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of Lloyd-type methods for the k-means problem. In *IEEE Symposium on Foundations of Computer Science*, 165–176, 2006.

[177] F. Pfender, G. M. Ziegler, G. Unter, and M. Ziegler. Kissing numbers, sphere packings, and some unexpected proofs. *Notices of the American Mathematical Society*, 51:873–883, 2004.

[178] S. Pion. Interval arithmetic: An efficient implementation and an application to computational geometry. In *Workshop on Applications of Interval Analysis to Systems and Control*, 99–110, 1999.

[179] S. Pion and M. Teillaud. 3D triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[180] S. Pion and M. Yvinec. 2D triangulation data structure. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

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

[182] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, 1990.

[183] R. Anderssen, R. Brent, D. Daley, and P. Moran. Concerning $\int_0^1 \ldots \int_0^1 \sqrt{x_1^2 + \ldots + x_k^2} dx_1 \ldots dx_k$ and a Taylor Series Method. *SIAM Journal on Applied Mathematics*, 30:22–30, 1976.

[184] R. J. Renka. Algorithm 772: STRIPACK: Delaunay triangulation and Voronoi diagram on the surface of a sphere. *ACM Transactions on Mathematical Software*, 23(3):416–434, 1997. Software available at http://orion.math.iastate.edu/burkardt/f_src/stripack/stripack.html.

[185] S. Rippa. Minimal roughness property of the Delaunay triangulation. *Computer Aided Geometric Design*, 7:489–497, 1990.

[186] G. Rote, F. Santos, and I. Streinu. Pseudo-triangulations — a survey. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry—Twenty Years Later*, volume 453 of *Contemporary Mathematics*, 343–410. American Mathematical Society, 2008.

[187] D. Russel. *Kinetic Data Structures in Practice.* PhD thesis, Stanford University, 2007.

[188] D. Russel, Menelaos I. Karavelas, and L. J. Guibas. A package for exact kinetic data structures and sweepline algorithms. *Computational Geometry: Theory and Applications*, 38(1-2):111–127, 2007.

[189] M. J. Sabin and R M Gray. Global convergence and empirical consistency of the generalized Lloyd algorithm. *IEEE Transactions on Information Theory*, 32(2):148–155, 1986.

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

[191] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[192] J. T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, 391–430. Elsevier, Amsterdam, 1990.

[193] R. Seidel. A convex hull algorithm optimal for point sets in even dimensions. M.Sc. thesis, Dept. Comp. Sci., Univ. British Columbia, Vancouver, BC, 1981. Report 81/14.

[194] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, 404–413, 1986.

[195] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991.

[196] M. I. Shamos. Geometric complexity. In *Proceedings of the 7th ACM Symposium on Theory of Computing*, 224–233, 1975.

[197] M. I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th IEEE Symposium on Foundations of Computer Science*, 151–162, 1975.

[198] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[199] M. Sharir and E. Yaniv. Randomized incremental construction of Delaunay diagrams: Theory and practice. Technical report, Tel Aviv Univ, Israel, 1990.

[200] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 14th ACM Annual Symposium on Computational Geometry*, 76–85, 1998.

[201] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, 1148:203–222, 1996.

[202] J. R. Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 16th ACM Annual Symposium on Computational Geometry*, 350–359, 2000.

[203] J. R. Shewchuk. Stabbing Delaunay tetrahedralizations. *Discrete and Computational Geometry*, 32:343, 2002.

[204] J. R. Shewchuk. Star splaying: an algorithm for repairing Delaunay triangulations and convex hulls. In *Proceedings of the 21th ACM Annual Symposium on Computational Geometry*, 237–246, 2005.

[205] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1978.

[206] R. Sibson. The Dirichlet tesselation as an aid in data analysis. *Scandinavian Journal of Statistics*, 7:14–20, 1980.

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

[208] S. Skyum. A sweepline algorithm for generalized Delaunay triangulations and a simple method for nearest-neighbour search. In *Abstracts of the 8th European Workshop on Computational Geometry*, 41–43, 1992.

[209] M. Smid. Closest point problems in computational geometry. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, 877–935. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[210] J. Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 30, 559–574. CRC Press LLC, Boca Raton, FL, 1997.

[211] B. Speckmann. Kinetic Data Structures for Collision Detection. PhD thesis, University of British Columbia, 2001.

[212] J. M. Steele. Growth rates of euclidean minimal spanning trees with power weighted edges. *Annals of Probability*, 16:1767–1787, 1988.

[213] J. M. Steele. Cost of sequential connection for points in space. *Operations Research Letters*, 8:137–142, 1989.

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

[215] K. Sugihara. Laguerre Voronoi diagram on the sphere. *Journal for Geometry and Graphics*, 6(1):69–81, 2002.

[216] J. Tournois, P. Alliez, and O. Devillers. Interleaving Delaunay refinement and optimization for 2D triangle mesh generation. In *Proceedings of the Meshing Roundtable*, 83–101, 2007.

[217] J. Tournois, C. Wormser, P. Alliez, and M. Desbrun. Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Transactions on Graphics*, 28(3), 2009.

[218] S. Valette, J. M. Chassery, and R. Prost. Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):369–381, 2008.

[219] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier Mémoire: Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1907.

[220] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st Annual Conference on Computer graphics and Interactive Techniques*, 269–277, 1994.

[221] 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*, 452–492. World Scientific, Singapore, 2nd edition, 1995.

[222] J. E. Yukich. Worst case asymptotics for some classical optimization problems. *Combinatorica*, 16(4):575–586, 1996.

[223] M. Yvinec. 2D triangulations. In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 3.6 edition, 2010.

[224] Y. Zhou, F. Sun, W. Wang, J. Wang, and C. Zhang Fast Updating of Delaunay Triangulation of Moving Points by Bi-cell Filtering. *Pacific Graphics*, to appear, 2010.

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA
COMMUNICATION

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

1. TITRE DE LA THESE : Méthodes pour Accélérer les Triangulations de Delaunay

2. NOM ET PRENOM DE L'AUTEUR : **Monsieur Machado Manhaes de Castro Pedro**

3. MEMBRES DU JURY :   *Günter Rote*
    Raphaelle Chaine
    Dominique Attali
    Olivier Devillers

4. PRESIDENT DU JURY :
    Bruno Lévy

5. DATE DE LA SOUTENANCE :
    25 Octobre 2010

6. REPRODUCTION DE LA THESE SOUTENUE :

    o   <u>Thèse pouvant être reproduite en l'état</u>

    o   Thèse ne pouvant pas être reproduite

    o   Thèse pouvant être reproduite après correction suggérées au cours de la soutenance

<u>Signature du Président du jury</u>

Ecole Doctorale STIC de l'Université de Nice-Sophia Antipolis