Robust and efficient implementation of the Delaunay tree

Olivier Devillers

Robust and efficient implementation of the Delaunay tree*

Implantation robuste et efficace de l'arbre de Delaunay

Olivier Devillers†

Programme 4 : Robotique, Image et Vision.

Rapport de recherche INRIA 1619, Février 1992.

†INRIA, 2004 Route des Lucioles, B.P.109, 06561 Valbonne cedex (France), Phone : +33 93 65 77 62, E-mail : Olivier.Devillers@alcor.inria.fr.

**This work has been supported in part by the ESPRIT Basic Research Action Nr. 3075 (ALCOM).*

Key words : Delaunay triangulation, Voronoi diagram, Robust algorithms, Randomized algorithms, Numerical precision, Degenerated cases.

Mots clés : Triangulation de Delaunay, Diagramme de Voronoï, Algorithmes robustes, Algorithmes randomisés, Précision numérique, Cas dégénérés.

Abstract

In this paper, we present some practical results concerning the implementation of the algorithm described in [DMT] which computes dynamically the Delaunay triangulation of a set of sites in the plane in logarithmic expected update time.

More precisely, we show that the hypotheses of non degenerate positions can be dropped and the problem of precision can be treated correctly.

Résumé

Cet article présente quelques résultats pratiques concernant l'implantation pratique de l'algorithme [DMT] qui calcule dynamiquement la triangulation de Delaunay d'un ensemble de points du plan en temps moyen de mise à jour logarithmique.

Plus précisément, nous montrons que les hypothèses de non dégénérescence peuvent être supprimées et les problèmes de précision numériques résolus correctement.

1 Introduction

Computational geometry is still a young disipline and by sometimes really theoritical and far from the applications. In parallel with an important research done on theoritical complexity of geometric algorithms, there is some work to achieve practical implementations [MN89,EKM*90,NSdL*91], all people who deal with practice must face the problem of numerical errors due to the use of rounded arithmetic.

Most of the algorithms used in computational geometry works on data belonging to a continuous universe, indeed the practical representation of this data in a computer is basically discrete. The most common attitude consists of ignoring this fact and the authors consider that they are allowed to use arithmetical operations for real numbers in the design of algorithms, and to suppose that these give correct results.

In practice, programers must use approximative methods to effectively code algorithms. One of the possibility is to use a "small" value ε and to consider that all smallest number are treated as null values and to use floating point arithmetic without more scrupules. But experience shows that this approach does not yield robust algorithms, in several problems, numerical errors introduce incoherences in the data structure and the algorithm fails. The problem comes from we have reasoned using Euclidean geometry and metric topology, but in a dicrete world where they are not valid.

To solve this problem, several attitudes are possible. As computers work naturally in a discrete world, we can define discrete objects and develop a discrete geometry to manipulate them, the properties of this geometry are often different to the Euclidean ones [Fra89,GY86]. Another approach consists of teaching the computer the Euclidean and continuous world, this kind of attitude begins with the use of rational arithmetic and continues with symbolic computation on an exact representation of some real numbers. Finally it is possible to exploit the floating point arithmetic, but with some evaluation of the error [For89,GSS89] to prove some robustness of algorithms [Mil89].

In this paper, our approach seems to be more related to the second one, because the geometric objects are not computed but stored in their symbolic description. The difference is we do not use symbolic computation on real numbers but directly on the special geometric objects our application deals with. We take a very practical point of view and we use the least possible symbolic computation. This approach can also have some similarity to the first one. To run an algorithm we need, for example, one Euclidean theorem which is not true in full generality in the discrete world of the computer ; so we just have to design our symbolic operations to make this theorem true (even if some other Euclidean theorems, not relevant here, are not verified).

We deal here, with precision problems, which is different to the problem of degenerate cases. One of the extensively used arguments in solving degenerate cases is the perturbation method [EM90,EC91], this method does not solve precision problems, and in fact increases them because the perturbed data has to be handled using more accurate precision than the original one.

2 The algorithm

The Delaunay tree is a data structure introduced in [BT86,BT,Tei91] to compute semi-dynamically the Delaunay triangulation of a set of sites in Euclidean d-dimensional space. This algorithm was extended in [DMT] to allow deletion as well as insertion. In this paper we deal only with the two dimensional case. The main result of these papers, is that the Delaunay tree allows the computation of the Delaunay triangulation of a set of n sites in general position in the plane in $O(\log n)$ expected insertion time and $O(\log \log n)$ expected deletion time. These bounds are randomized, i.e. all possible orders for already inserted sites are supposed to be equally likely, and when a site is deleted, it may be any site with the same probability.

We refer the interested reader to these papers for the details of the algorithm. And we only sketch the most relevant features of the Delaunay tree.

The Delaunay tree is in fact a directed acyclic graph whose nodes are triangles. A site p is said to be in conflict with a triangle T if $p \in B(T)$ the interior of the ball circumscribing T. So the Delaunay triangulation is the set of triangles without conflicts. To handle the edges of the convex hull in simple way, we can consider them as *infinites* triangles whose circumscribing circle has an infinite radius, so the ball $B(E)$ corresponding to an edge of the convex hull E is the half plane limited by the line supporting E.

The Delaunay tree must verify two fundamental properties :

1. The ball of a triangle S is included in the union of the balls of its two parents T and R : $B(S) \subset B(T) \cup B(R)$.

2. The triangles without conflict are present in the Delaunay tree.

Adding a site

When a new site p is added, Property 1 allows the efficient determination of the triangles in conflict with p by a simple graph traversal. Then, for every Delaunay triangle T in conflict with p whose neighbor R is not, we create a new triangle S by linking p to the common edge of T and R. In the Delaunay tree S is made child of T and R (see Figure 1). Triangle T, which is no longer a Delaunay

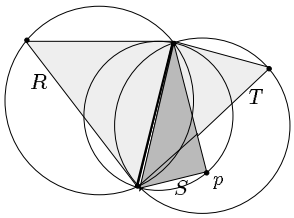

Figure 1: Creation of a new triangle

one, remains in the tree, thus the Delaunay tree keeps trace of all the successive Delaunay triangulations.

Deleting a site

We now want to remove a site p of Delaunay tree. All the triangles incident to p must be removed : some of them are triangles of the current Delaunay triangulation but other ones are not ; they correspond to internal nodes of the Delaunay Tree, and must be removed, too. Moreover, we must restore the Delaunay Tree in the same state it would be in if p had never been inserted, and if the other sites had been inserted in the same order. The basic idea of the algorithm is to reinsert the sites inserted after and near p, but only to replace the removed triangles locally.

Geometry

As any non trivial algorithm, ours uses some geometrical properties. This is the main problem of implementing geometric algorithms : some **Euclidean** geometrical properties are used and are not necessarily verified in the *rounded* geometry of the computer world.

We need to clearly identify the geometrical properties used by our algorithm and to make sure they are still true after discretization by the use of rounded arithmetic.

Here, the basic numerical operation is testing if a point lies inside or outside a circle and the fundamental property is that when a new triangle is created the inclusion property is verified. By verifying the inclusion property, we mean that if, for some site q, the "rounded" test $q \in B(S)$ is positive, it must be positive also for at least one of the two "rounded" tests $q \in B(T)$ and $q \in B(R)$.

Another property is that for every point in the plane, there exist at least one triangle in conflict with this point.

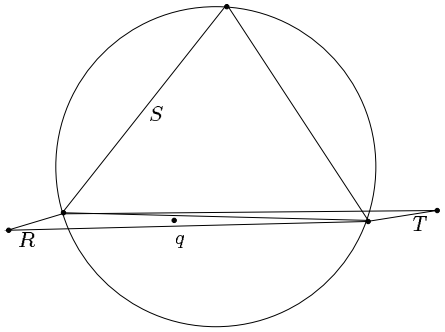

Figure 2: $B(S) \subset B(T) \cup B(R)$ may be false for the computer

3 Towards a practical implementation

When implementing the algorithm above, precision problems may occur. For example, on Figure 2, five sites are almost colinear and it is possible that the rounded arithmetic test say that $q \notin B(T)$, $q \notin B(R)$ and $q \in B(S)$ although the algorithm supposes that $B(S) \subset B(T) \cup B(R)$. The property of inclusion of the disks would be true in Euclidean geometry but unfortunatly we are in a discrete world.

Another problem occur with degeneracies. In our problem we call a degenerate configuration a set of sites where more than four sites are cocircular (or more than three are colinear). One of the common arguments for not dealing with degeneracies is that such a configuration occurs with probability 0, which is only true in a continuous set with some continuous probability laws, but in a discrete context probability must be discrete also and the probability of a degenerate case, even if it is very low, is not zero. In a discrete computational geometry, the degenerate cases must be dealt with.

Exact integer arithmetic

In the sequel, to avoid precision problems, exact computations are used. It is plain to observe that if two integers x and y are stored using b bits (i.e. $-2^{b-1} \leq x, y < 2^{b-1}$) then $b+1$ (*resp.* $2b$) bits are enough to store the exact value of the sum $x+y$ (*resp.* the product xy).

When the usual integer arithmetic provided by a computer, all the computations are done exactly provided that the precision allowed by the computer is not exceeded, for example an integer is often stored using 32 bits. But, we have to notice that usual rounded floating point arithmetic can produce a better result, more precisely, while the computation can be done exactly no approximation is

done, so if the number of bits used on a given computer to store the mantissa of a real number is larger than the number of bits used to store an integer, you get a better integer arithmetic by using the floating point arithmetic.

In the sequel, the aim is to do all computations exactly but to use as much as possible the usual capabilities of the arithmetic of the computer.

3.1 Solving the precision problem

If we look at our algorithm carefully, we see than the only numerical calculations, and the only posibility of precision problem, is the test $s \in B(T)$ for a point s and a triangle T.

So the idea is simple, we compute this test exactly.

Suppose that the vertices of T are p, q, r given in counter-clockwise order by their coordinates $x_p, y_p, x_q, y_q, x_r, y_r$ and s also by their coordinates x_s, y_s. The test can be done by computing the following determinant :

$$\delta = \begin{vmatrix} x_s - x_p & x_q - x_p & x_r - x_p \\ y_s - y_p & y_q - y_p & y_r - y_p \\ x_s^2 + y_s^2 - x_p^2 - y_p^2 & x_q^2 + y_q^2 - x_p^2 - y_p^2 & x_r^2 + y_r^2 - x_p^2 - y_p^2 \end{vmatrix}$$

- If $\delta < 0$ then s is inside $B(T)$,
- if $\delta = 0$ then p, q, r and s are cocircular (degenerate case) and
- if $\delta > 0$ s is outside $B(T)$.

We can developped δ according to the last line :

$$\begin{aligned} \delta \;=\; & (x_s^2 + y_s^2 - x_p^2 - y_p^2) \begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix} \\ & + (x_q^2 + y_q^2 - x_p^2 - y_p^2) \begin{vmatrix} x_r - x_p & x_s - x_p \\ y_r - y_p & y_s - y_p \end{vmatrix} + (x_r^2 + y_r^2 - x_p^2 - y_p^2) \begin{vmatrix} x_s - x_p & x_q - x_p \\ y_s - y_p & y_q - y_p \end{vmatrix} \end{aligned}$$

As claimed before, our aim is to compute the exact value of δ, without any rounded operations.

If we suppose that the coordinates of the sites are integers stored using b bits then the computation of this determinant needs to use an exact integer arithmetic with a precision of $4b + 6$ bits.

But, if we look more in the details of the calculation, δ has the form $a\alpha + b\beta + c\gamma$ where a, b, c, α, β and γ need only a precision of $2b+2$ bits and the last expression need a precision of $4b + 6$ bits to be evaluated correctly.

At this time, we have to particularize our computer. Assume you use 32 bits integers, the evaluation of a, b, c, α, β and γ can be done using the standard arithmetic of the machine provided that $2b+2 \leq 32$. Now, you need the maximum

of the precision available on your computer for the final evaluation of δ, for example if you have a specialized processor for floating point arithmetic with some registers whose mantissa is 64 bits long thus you can use it for the last calulous. In this way you must have $4b + 6 \leq 64$, so $b \leq 14$ and finally the sites must have integer coordinates between -8192 and 8192.

If this precision is not sufficient for your needs, you can use some library for precise integers, but the important point is that you only need to use it for the evaluation of a single expression and not in the whole program.

3.2 Degenerate cases

As noticed above, we must solve the degenerate cases, it is no more possible in this context to consider they do not happen. The perturbation methods [EM90,EC91] are not acceptable too, this method introduce a small perturbation of the data which ensures that degenerate cases no longer occur, but this method increase considerably the precision needed on the data. This is not desirable, by virtue of the considerations of the preceding section.

In the description of the algorithm in Section 2, a general position assumption is made, but in fact, we can remove it using a new definition of $B(T)$ as long as the inclusion of balls property is verified.

We now define $B(T)$ as the closed sphere circumscribing T. With this new definition, it is easy to verify that the inclusion property is verified even for cocircular sites. The only problem may occur if we need to insert exactly the same site for a second time, but this case can be easily detected, and as the site is already present there is no need for a second insertion.

This approach solves the first case of degeneracy, when four points are cocircular. The case of three colinear points introduces only some particular cases, easy to handle. When three points are detected to be exactly colinear, then the ball circumscribing the triangle formed by these three points is one of the half plane delimited by the line supporting the three points. In fact, the test proposed above to test if a point p is inside or outside $B(T)$ works well even if T is such a degenerate triangle.

4 Practical results

This algorithm has been effectively coded and can be compared favorably with the best known implementations. Furthermore, this algorithm has the advantage of being dynamic (insertions and deletions) which is not the case of divide and conquer or sweeping ones. To give an idea of the performances, the Voronoi diagram of 15000 random sites in a square have been computed in 30 seconds and removed (in a random order different from the order of insertion) in 60

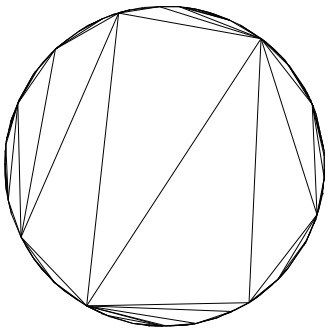

Figure 3: Delaunay triangulation of 200 random sites on a circle

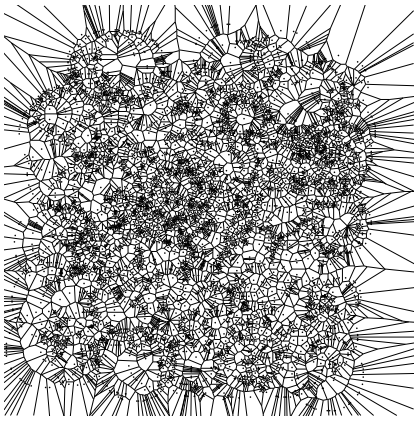

Figure 4: Voronoi diagram of 5000 sites on 200 random circles

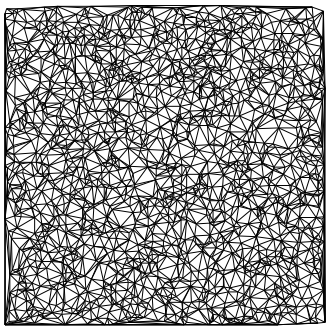

Figure 5: Delaunay triangulation of 2000 random sites in a square

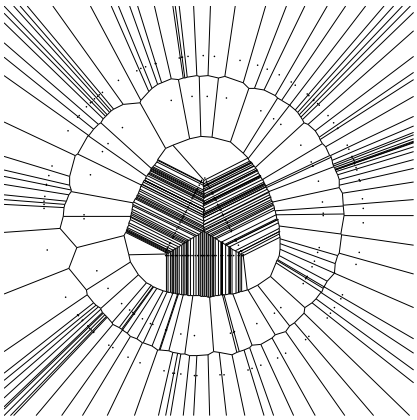

Figure 6: Voronoi diagram of 300 random sites in a degenerate configuration

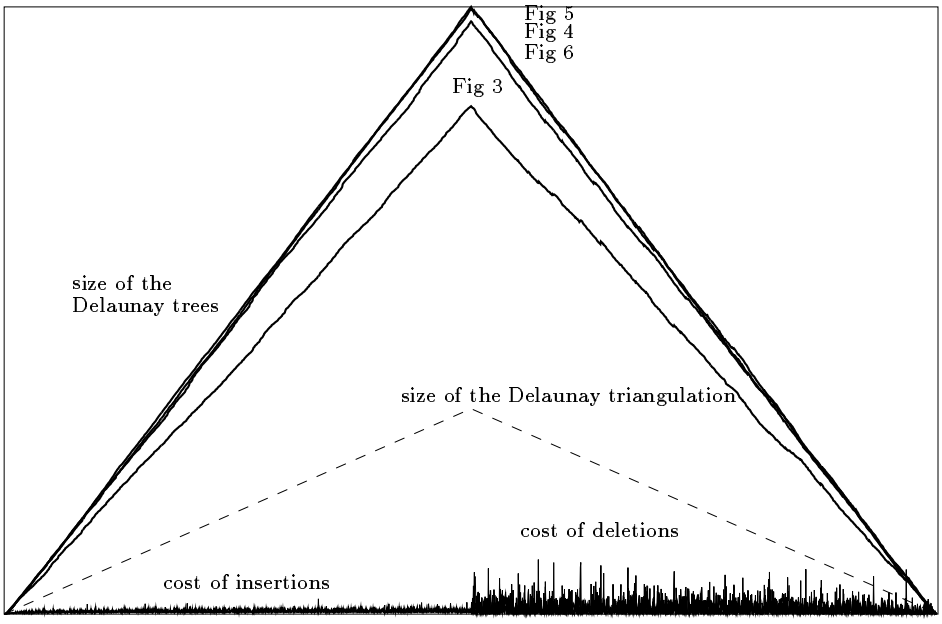

Figure 7: Statistics on 1000 sites in different positions

seconds, this time are for a SUN 4/75. This software is available in the LEDA library [MN89].

Figures 3 to 6 show some Delaunay triangulations or Voronoi diagrams of set of points. Some of them are fairly degenerate and are handled without problem by our software.

The program has been used for data sets sinilar to those of Figures 3 to 6, but to compare the results all the sets have size 1000. The sites are first inserted in a random order, and afterwards they are all deleted in another random order. Figure 7 presents practical results : the size of the Delaunay Tree in bold line, the size of the Delaunay triangulation in dashed line, and in thin line, a measure of the complexity of the operation. For insertions, it is the number of visited nodes, for deletions it is the number of unhooked triangles plus the numbers of triangles created during this deletion plus the cost of sorting the points to be reinserted. It is important to notice than these results does not depend significantly on the different data.

5 Conclusion

The kind of approach developped in this paper for Delaunay triangulation can be used for most of the fundamental structures in computational geometry. The important things are that all the numerical computations are based on the original definition of the geometric objects, the computation are done using exact arithmetic and the degenerate cases are treated apart. It is necessary to attack each problem separately to describe the data in an appropriate manner.

The standard perturbation methods present the disadvantage of drastically increasing the precision needed to store the data. The way of formalizing the method described above would be the development of a discrete geometry which handles the dicrete nature of the computer. As the properties of this geometry would not use real arithmetic but a discrete one, they would be really exploitable in the algorithms.

References

[BT] J.D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science.* To appear. Available as Technical Report INRIA 1140.

[BT86] J.D. Boissonnat and M. Teillaud. A hierarchical representation of objects: the Delaunay Tree. In *Second ACM Symposium on Computational Geometry in Yorktown Heights*, June 1986.

[DMT] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry Theory and Applications*. To appear. Available as Technical Report INRIA 1349. Abstract published in LNCS 519 (WADS'91, august 1991).

[EC91] I. Emiris and J. Canny. A general approach to removing degeneracies. 1991. Manuscript, University of California, Berkeley.

[EKM*90] P. Eppstein, A. Knight, J. May, T. Nguyen, and J-R. Sack. *A workbench for computational geometry.* Technical Report SCS-TR-180, School of computer science, Carleton University, September 1990.

[EM90] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, January 1990.

[For89] S. J. Fortune. Stable maintenance of point set triangulations in two dimensions. In *IEEE Symposium on Foundations of Computer Science*, pages 494–499, 1989.

[Fra89] J. Françon. Topologie discrète et algorithmique graphique. *Bigre-Globule*, 61:91–96, April 1989. Journées AFCET-GROPLAN de Décembre 1988 à Toulouse.

[GSS89] L.J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry : building robust algorithms from imprecise computations. In *5th ACM Symposium on Computational Geometry in Saarbrücken*, pages 208–217, June 1989.

[GY86] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *IEEE Symposium on Foundations of Computer Science*, pages 143–152, 1986.

[Mil89] V. Milenkovic. Double precision geometry : a general technique for calculating line and segment intersections using rounded arithmetic. In *IEEE Symposium on Foundations of Computer Science*, pages 500–505, 1989.

[MN89] K. Mehlhorn and S. Näher. *LEDA, a library of efficient data types and algorithms.* Technical Report TR A 04/89, Universität des Saarlandes, Saarbrücken, 1989.

[NSdL*91] J. Nievergelt, P. Schorn, M. de Lorenzi, C. Ammann, and A. Brüngger. *XYZ : Software for geometric computation.* Technical Report 163, Institut für Theorische Informatik, ETH, Zurich, July 1991.

[Tei91] M. Teillaud. *Vers des algorithmes randomisés dynamiques en géométrie algorithmique.* December 1991. Thèse de Doctorat en Sciences, Université Paris-Sud, (France).