Fast Delaunay point location with search structures

Luc Devroye* Christophe Lemaire† Jean-Michel Moreau‡

Extended abstract – submission to *CCCG'99*

Note: *this extended abstract is the shorter version of a full paper ([4]) with same title, submitted to this Conference, the postscript version of which may be retrieved at the address given in the bibliography.*

1 Introduction

We study the expected time behaviour of the Jump-and-walk paradigm when the set of sites is controlled by a binary search tree or a well-balanced 2-d tree. Throughout the paper, we shall assume that we are given a Delaunay triangulation on N sites *uniformly distributed* in the unit 2-dimensional square, $[0,1]^2$. We are requested to locate a query point q, which will be assumed to be bounded away from the boundary of the Delaunay triangulation, as the expected analysis of the general case calls for more powerful tools (to be published in a future paper).

The rest of the paper is organized as follows: Section 2 presents *(oversampled) BinSearch & Walk*, an optimization of *Jump & Walk*, and analyzes its expected running time. We then analyze the expected running time of a static variant of this scheme, based on a 2-d tree (Section 3). Experimental and comparative results are given in Section 4, and we conclude on possible extensions and generalizations.

*Research sponsored by NSERC Grant A3456 and by FCAR Grant 90-ER-0291. Address: School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2K6. Email: luc@cs.mcgill.ca.

†The research of the second author was done while he was at SETRA. New address: CEA / MLS BP 12, 91680 Bruyères-le-Châtel, France. Email: lemairec@bruyeres.cea.fr.

‡Research partially supported by LNH-EDF, Chatou, France. Address: LISSE / ENSM.SE, 158 Cours Fauriel, St-Etienne, France. Email: Jean-Michel.MOREAU@emse.fr.

2 Using a balanced 1-d tree

Jump & Walk ([6]) proceeds as follows: pick $k \in [1,N]$ random sites from the data, select ζ, the one closest to q, and traverse the triangulation from ζ to q, exploiting the adjacency relationships between the successive triangles crossed by segment (ζ, q). Using the following lemma:

Lemma 2.1 (Bose & Devroye [2]) *Let $\epsilon > 0$ be fixed, and let $\mathcal{L}$ be a random line segment on $[\epsilon, 1-\epsilon]^2$. Then the expected number of triangles and edges of a Delaunay triangulation for N random sites on $[0,1]^2$ cut by $\mathcal{L}$ is bounded from above by $c_0 + c_1|\mathcal{L}|\sqrt{N}$, where c_0 and c_1 are universal positive constants not depending upon $\mathcal{L}$ or N.*

Devroye, Mücke & Zhu proved that the expected running time of their method is $O(k + \sqrt{N/k})$, which reaches its optimum ($O(N^{1/3})$) when k is $\Theta(N^{1/3})$. If we now assume that the sites are organized as a dynamically maintained weight-balanced binary search tree (bounded balanced trees, *cf.* [1]), *BinSearch & Walk* proceeds as follows: start from the root of the tree, search for q; while doing so, determine, among all sites visited, the one, ζ, closest to q. Finally walk from ζ to q, as for *Jump & Walk*. We can prove the following Theorem:

Theorem 2.2 *The expected running time of* Bin-Search & Walk *is* $O(\sqrt{N}/\log N)$.

Sketch of proof: In a bounded balanced tree, we are given a constant $\alpha \in [1/4, 1-1/\sqrt{2}]$, and the following property: if ν is any node in the tree with left subtree $l(\nu)$, and we denote by $\sigma(\nu), \sigma(l(\nu))$ the number of external nodes (leaves) in ν and $l(\nu)$, respectively, the ratio $\sigma(l(\nu))/\sigma(\nu)$ is always in $[\alpha, 1-\alpha]$. (Note that this property also applies to ν's right subtree.) A direct consequence is that all the root-to-leaf paths in such a tree on N sites contain a subtree with $\Theta(\sqrt{N})$ nodes. We now consider that the k sites on the remaining part of the search paths are the samples to initiate the walk. Using well-known properties of order statistic variables with unit uniform parent, the expected length of the horizontal side of the rectangle containing q and all sites carried by the above mentioned subtree is $\Theta(1/\sqrt{N})$. Next, we can show that, the vertical distance between q and its nearest neighbour in this rectangle is $O(1/k)$. Using the triangle inequality and Lemma 2.1, the overall time for the method is $O(\log N + k + \sqrt{N}/k)$, accounting for searching, sampling, and walking. The theorem follows from the fact that k is $\Theta(\log N)$, since any $\mathbb{B}_\alpha$-tree on M nodes has $\Theta(\log M)$ height. □

Since it is possible to perform insertions (and deletions) at logarithmic cost per operation on such bounded balance trees, the location algorithm may be used in a dynamic context. Obviously, its worst case performance is linear in N.

Observe that the optimum of $O(k + \sqrt{N}/k)$ is $O(N^{1/4})$, and is reached when k is $\Theta(N^{1/4})$. Hence, we can optimize the previous method by *oversampling*: traverse the tree from root to leaf, then move back up the search path, recursively traversing all non-visited subtrees (while still registering the visited site closest to q), until we have "sampled" $\Theta(N^{1/4})$ nodes. This yields:

Theorem 2.3 *The expected running time of* Oversampled-BinSearch & Walk *on N random sites uniformly distributed in a square is $O(N^{1/4})$ (provided the query point is bounded away from the boundary of the triangulation).*

A more formal presentation of *Oversampled Bin-Search & Walk* and of its analysis may be found in [5]. Note that $N \to N^{1/4}$ is competitive with $N \to \log_2 N$ up to more than 10,000,000 sites.

3 Using a balanced 2-d tree

If we remove the possibility of inserting or removing sites, we may devise an even faster method, as follows. The unit square is recursively partitioned by a perfectly balanced 2-d tree, that is, the partitioning alternates directions between x and y, and member sets in the partition are rectangles. Every node in the 2-d tree receives one data point, about which the remaining points are split. Leaves correspond to empty rectangles. It is clear, therefore, that there are N non-leaf nodes, and $N+1$ leaves. For a rectangle that properly contains a collection of data points, if a vertical split is made, it is made at the x-median of these points, where the x-median is uniquely defined if the number of points is odd, and is the leftmost of the two candidate medians when N is even. Horizontal splits are made about y-medians. A rectangle with one point is thus split about this point, and this results in two empty leaf rectangles.

When splits alternate between horizontal and vertical, we end up with a rather balanced 2-d tree. This structure is used in a static manner then for point location in a Delaunay triangulation for $s_1, \ldots, s_n$. Assume a query point q is given. We determine in $O(\log_2 n)$ worst-case time the leaf region for q. Then we let ζ be the node at the parent of this leaf. This node's position in the Delaunay triangulation is of course known (in $O(1)$ time). Then walk across the segment (ζ, q) in the Delaunay triangulation to determine the triangle for q. The extra time needed for this walk is $O(1)$ plus the number of Delaunay edges crossed by the walk. If q is restricted as before, we can prove the the following Lemma:

Lemma 3.1 $\mathbf{E}\{\|\zeta - q\|\} \leq e^{\sqrt{8}}\sqrt{8/n}$.

Proof: Omitted in this extended abstract. See full paper ([4]). □

In other words, the expected length of the longest side in any leaf rectangle determined by the static 2-d tree is $O(1/\sqrt{N})$, and the expected running time for the walk phase is $O(1)$ (assuming the above mentioned restriction on q). Hence, the overall expected time to locate q among N random sites is $O(\log N)$.

4 Experiments

The full paper presents experimental results, and comparative tests with various methods, including O. Devillers' randomized location algorithm ([3]).

5 Conclusion

We have shown how to use simple order-based search structures to locate query points in a Delaunay triangulation. Such structures are also very useful in the construction of the Delaunay triangulations. The combination of the two strategies allows hybrid applications: first build the massive triangulations using an order-based optimal algorithm, and then perform the relatively fewer location operations. Actually, *BinSearch & Walk* is used in a real-scale constrained Delaunay mesh application developed at ENSM.SE for (and with the financial support of) the French Board of Electricity (EDF-LNH, Chatou, France).

We are currently investigating the generalization of the methods presented in this paper to divided k-d trees, a powerful dynamic structure for multi-dimensional range searching ([8]), and working on the expected time analysis of all these methods in the general, unrestricted, case.

References

[1] N. Blum and K. Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theoretical Computer Science*, 11:303–320, 1980.

[2] P. Bose and L. Devroye. Intersections with random geometric objets. *Computational Geometry: Theory and Applications*, 10:139–154, 1998.

[3] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proceedings of the 14th Annu. ACM Symp. Comput. Geom.*, pages 106–115, Minneapolis, Minnesota, May 1998. Also *Technical Report 3298*, INRIA, Sophia Antipolis, France, November 1997.

[4] L. Devroye, C. Lemaire, and J-M. Moreau. Fast Delaunay point location with search structures. Full paper version. Submitted to *CCCG'99*. Postscript version available at `http://www.emse.fr/ECOLE/FRENCH/SIMADE/LISSE/FTP/dlm-cccg-fp-1999.ps.gz`.

[5] L. Devroye, C. Lemaire, and J-M. Moreau. Fast Delaunay point location with search structures. Technical report, LISSE-ENSM.SE, St-Etienne, to appear.

[6] L. Devroye, E. P. Mücke, and B. Zhu. A note on point location in Delaunay triangulations of random points. See also [7], 1995.

[7] 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 Annu. ACM Symp. Comput. Geom.*, pages 274–283, Philadelphia, 1996.

[8] M. J. van Kreveld and M. H. Overmars. Divided k-d trees. *Algorithmica*, 6:840–858, 1991.