Delaunay triangulation programs on surface data[*]

Sunghee Choi[†] Nina Amenta[‡]

Abstract

The Delaunay triangulation of a set of points in 3D can have size $\Theta(n^2)$ in the worst case, but this is rarely if ever observed in practice. We compare three production-quality Delaunay triangulation programs on some 'real-world' sets of points lying on or near 2D surfaces.

1 Introduction

Developing good programs for 3D Delaunay triangulation is a difficult and important part of computational geometry. Two new codes, the Delaunay function of Shewchuk's `pyramid` and `Delaunay hierarchy` function of the `CGAL` library [1], represent a new standard of quality. We examine their performance, and that of an older program, Clarkson's `hull`, on sets of points which lie on or near two-dimensional surfaces in R^3. This special case is important in applications such as reconstructing surfaces from point clouds and and meshing three-dimensional solids.

Our first goal is to provide some formal evidence for the 'folklore' observation that such Delaunay triangulations have linear size. This observation justifies their use in practice and has prompted recent theoretical results: the Delaunay triangulation of random points on a convex polytope has linear size [7], for a 'well-sampled' fixed smooth surface there is an upper bound of $O(n^{7/4})$ [2], and for any n, there is a 'bad surface' which is 'well-sampled' but gives a quadratic Delaunay triangulation [6].

Our second goal is to identify the bottlenecks in current implementations. Contrary to our expectations, we find that the point location subroutine does not dominate the running time on these practical examples, and instead the most immediate challenge seems to be thrashing. Pion [9] offers some evidence that "good" insertion orderings can alleviate thrashing.

The programs: The three programs all use the optimal and easily implemented randomized incremental algorithm [3], which adds points one by one in random order while maintaining the Delaunay triangulation. If we assume that the Delaunay triangulation is always of linear size, then *point location* (finding where to add a new point p) is the only operation that is not $O(1)$ expected time per insertion. `Hull` uses a theoretically optimal $O(\lg n)$ data structure [3], which is memory intensive. `Pyramid` uses a simple $O(n^{1/4})$ *jump-and-walk* strategy [8]. `CGAL Delaunay hierarchy` uses a few levels of intermediate Delaunay triangulations [4] as a search structure, something like a skip list. This is known to be optimal in 2D, but in 3D, there is no theoretical result.

Datasets and platform: The `dragon` data (1.7 million points) is from the Stanford 3D Scanning Repository. It comes from a Cyberware range scanner; it contains noise and is very non-uniform. `MTD` (185,000 points) is from a protein electron density iso-surface, selected via marching cubes. `B-1` (525,000 points) and `B-2` (2 million points) were obtained by applying butterfly subdivision to such an iso-surface, once and twice, respectively, to get a smooth and dense point set. Experiments on other data sets showed similar results and can be seen at `www.cs.utexas.edu/users/sunghee/delaunay`. Timing does not include file I/O and all experiments are done in Linux on an Intel Pentium III (864 MHz) with 511M RAM.

2 Results

All of our datasets produced linear-sized Delaunay triangulations and the size of all intermediate Delaunay triangulations was linear. The number of Delaunay tetrahedra created/destroyed per insertion averaged about 27/20 and the average ratio of the number of Delaunay tetrahedra to the number of input points is about 6-7. See Figure 1.

The shape of the overall performance profile of all the programs was similar: near-linear running time until memory is exceeded, at which point thrashing occurs. See Figure 2. It took about 400 seconds for `CGAL Delaunay hierarchy` and about 350 seconds for `pyramid` to compute the Delaunay triangulation for a million points.

Point location time forms a significant, but not overwhelming fraction of the overall running time. See

[*]Supported by NSF:CCR-9731977 and an Alfred P. Sloan Foundation Fellowship.

[†]Department of Computer Sciences, The University of Texas at Austin

[‡]Department of Computer Sciences, The University of Texas at Austin

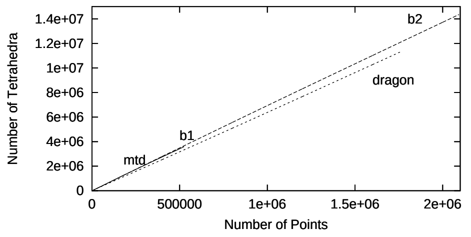

Figure 1: The number of Delaunay tetrahedra per number of input points added.

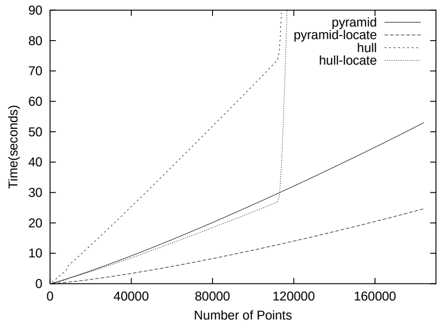

Figure 3: Total point location time vs. the total time for `hull` and `pyramid` on `MTD` data

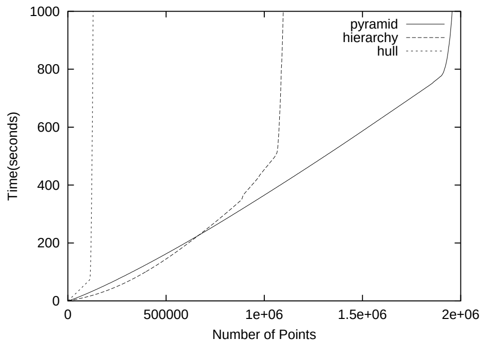

Figure 2: `Hull`, `CGAL Delaunay hierarchy`, and `pyramid` begin thrashing around 120,000, 1,000,000, and 1,800,000 points, respectively. `Hull` timing for `MTD` data, `CGAL Delaunay hierarchy/pyramid` for `B-2` data.

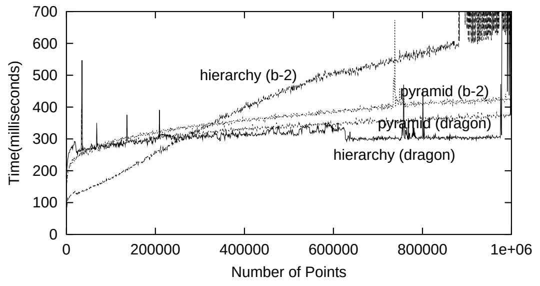

Figure 4: Time required per 1000 insertions, on `dragon` and `B-2` data, for `pyramid` and `CGAL Delaunay hierarchy` before they thrash.

Figure 3.

Time per vertex insertion was consistent for `pyramid` but varied for `CGAL Delaunay hierarchy`. `CGAL Delaunay hierarchy` has many options for arithmetic optimization; different choices were best for different datasets. See Figure 4.

Acknowledgments: We are very grateful to Clarkson for `hull`, to Shewchuk for a pre-release copy of `pyramid`, and to Teillaud and Pion for a pre-release version of `CGAL Delaunay hierarchy`.

References

[1] The CGAL website. `www.cgal.org`

[2] D. Attali and J-D. Boissonnat. Complexity of the Delaunay triangulation of points on a smooth surface. Manuscript, (2001).

[3] K. Clarkson, K. Mehlhorn and R. Seidel. Four results on randomized incremental constructions. *Proc. of the 9th Symposium on Theoretical Aspects of Computer Science*, (1992).

[4] O. Devillers. Improved incremental randomized Delaunay triangulation. *Proc. of the 16th Symposium on Computational Geometry*, 106-115, (1998).

[5] R. A. Dwyer Higher-dimensional Voronoi diagrams in linear expected time. *Discrete and Computational Geometry* 6:343–367, (1991).

[6] J. Erickson. Nice point sets can have nasty Delaunay triangulations. *Proc. of the 15th ACM Symposium on Computational Geometry*, (2001).

[7] M. Golin and H. Na. On the average complexity of 3D-Voronoi diagrams of random points on convex polytopes. *Proc. 12th Canadian Conference on Computational Geometry*, (2000), pp 127–135.

[8] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Proc. 12th ACM Symposium on Computational Geometry*, (1996)

[9] S. Pion. Personal communication.