Detri 2.2:
A Robust Implementation for 3D Delaunay Triangulations

(Extended Abstract)

E. P. Mücke
Computer Research & Applications Group (CIC-3)
Los Alamos National Laboratory, Mail Stop M986
Los Alamos, NM 87545
mucke@lanl.gov

Abstract: Detri 2.2 computes Delaunay triangulations of 3-dimensional point sets. The code uses a variant of the randomized incremental-flip algorithm, employing a symbolic perturbation scheme to achieve robustness. Its optimal worst-case time complexity is $O(n^2)$, where n is the number of input points. However, in praxis, we observe a run-time roughly proportional to the number of triangles in the final triangulation, which is typically linear in n.

Delaunay Triangulations

A *triangulation* $\mathcal{T}$ of a point set S of n points is a collection of nonoverlapping tetrahedra covering the convex hull of S; each point of S is endpoint of at least one tetrahedron in $\mathcal{T}$. The triangulation is called a *Delaunay triangulation*, denoted by $\mathcal{D}$, if the circumcircle of each tetrahedron *abcd* contains no point of S other than its endpoints a, b, c, and d. Delaunay triangulations are dual to *Voronoi diagrams*, and both concepts have a long history [17, 18] as well as a variety of applications in different disciplines [1]. In particular, 3D Delaunay triangulations are useful in the practical area of mesh or grid generation, and a robust and efficient implementation is therefore in high demand.

A classical lemma by Delaunay [3] implies that a triangulation is (globally) Delaunay iff each face is *locally Delaunay*. In 3D, a triangle *abc* is locally Delaunay if it either lies on the convex hull, or is shared by two incident tetrahedra *abcd* and *abce* and the circumsphere of *abcd* does not contain e. This local characterization of $\mathcal{D}$ is the basis for the idea to construct $\mathcal{D}$ from *some* triangulation $\mathcal{T}$ by a sequence of local tests, so-called *in-sphere* tests, and local transformations, so-called *flips*.

Lawson [13] was the first to do so, but only for the 2D case. In the plane, it is not too hard to see that an arbitrary sequence of "edge flips" will always transform an arbitrary triangulation into the Delaunay triangulation after $O(n^2)$ flips. Although this does not yield an optimal algorithm[1] it is by far the easiest to implement; moreover, the idea of flipping can be used to prove several optimality properties of $\mathcal{D}$. Unfortunately, a naïve generalization of flipping from 2D to 3D does not work. Starting with an arbitrary triangulation, and using an arbitrary sequence of flips, one can obtain a triangulation with triangles that are not locally Delaunay, but cannot be flipped. This was first reported by Joe [11]. However, he also came up with an alternative procedure, the "incremental flip," which always works [12].

Algorithm

The idea of the incremental flip algorithm is the following. Start with 4 points, and connect them to a tetrahedron; obviously this is the Delaunay triangulation of these 4 points, call it $\mathcal{D}_4$. Now, for i from 5 to n perform the following iterative step: assume $\mathcal{D}_{i-1}$ to be the Delaunay triangulation of the first $i-1$ points; add point i to it, forming a new triangulation $\mathcal{T}_i$ which is no longer necessarily Delaunay; transform $\mathcal{T}_i$ into $\mathcal{D}_i$ via a sequence of flips. It can be shown that this procedure is always successful [12] with an arbitrary sequence of flips[2] and a time complexity of $O(n^2)$, which is worst-case optimal.

The algorithm proposed in [12] sorts the points along the x-axis and incrementally adds them in this order. The advantage is that adding point i to $\mathcal{D}_{i-1}$, and thus forming $\mathcal{T}_i$, becomes very easy.[3] Unfortunately, this has the tendency to produce intermediate triangulations of

[1] In 2D, $\mathcal{D}$ can be computed in $O(n \log n)$ time [10, 9]

[2] Rajan [15], and also [7], show that incremental flipping works in any dimension, even for regular triangulations.

[3] Hint: point i will always lie outside $\mathcal{D}_{i-1}$, and point $i-1$ lies on the boundary of $\mathcal{D}_{i-1}$ and will always be visible from i.

higher complexity than the final triangulation, especially, for degenerate data, say, for points in planes perpendicular to the x-axis.

Randomizing the algorithm overcomes this: we follow the same procedure as above, only that now points are added in a random order. Clearly, we can no longer rely on point i lying outside $\mathcal{D}_{i-1}$, and if i lies inside, the tetrahedron containing it needs to be identified. Edelsbrunner and Shah [7] solved this point location problem by using a dag (directed acyclic graph) to store the history of all flips performed. This history dag serves as the data structure for point location, and results in an algorithm which is elegant and efficient in the expected case. Still, certain drawbacks remain, one of them is that the dag needs $O(|\mathcal{D}|) = O(n^2)$ additional storage.

The Detri 2.2 implementation uses a different method to locate point i in $\mathcal{D}_{i-1}$. It is simple: start with a random tetrahedron in $\mathcal{D}_{i-1}$; if it does not contain i, iterate the search with one of the incident tetrahedra that is "closer." This method of "walking through" the triangulation was already mentioned in [2] and, although, it is obviously bad in worst-case, it performs amazingly well in praxis, especially when one uses some heuristics to find a relatively "good" initial tetrahedron. For details see [14] where empirical evidence is given that the average search length is roughly $\frac{1}{2}$ of the average search length of the history dag method, and with only $O(1)$ additional space.

Robustness

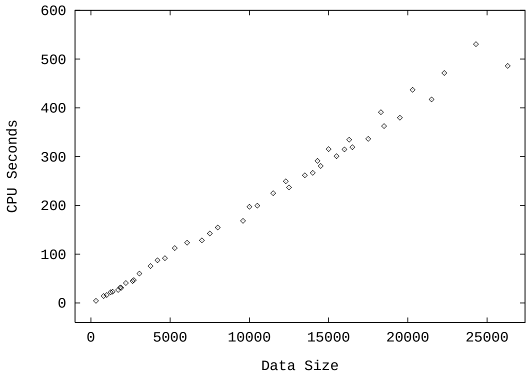
The code is robust in the following sense: if S is in general position, i.e., nondegenerate, then $\mathcal{D}$ is unique and Detri will compute it; if S contains degeneracies, then the computed $\mathcal{D}$ will be the Delaunay triangulation for a slightly perturbed point set $S(\varepsilon)$. This is achieved by a symbolic perturbation method called SoS, Simulation of Simplicity [5].

In a nutshell, SoS works as follows. For each point $p_i \in S$, its coordinates $\pi_{i,j}$ are replaced by polynomials in ε, namely $\pi_{i,j}(\varepsilon) = \pi_{i,j} + \varepsilon(i,j)$ where ε is assumed to be arbitrarily small but positive. The $\varepsilon(i,j)$ terms are chosen in such a way that any subdeterminant of the perturbed "parameter matrix" $(\pi_{i,j}(\varepsilon))$ is nonzero; in other words, $S(\varepsilon)$ is nondegenerate. Details can be found in [5, 14]; here only so much: SoS effectively replaces the evaluation of any determinant in the original coordinates, call it Δ, by an evaluation of polynomials in ε, $\Delta(\varepsilon)$, whose coefficients are subdeterminants of Δ. For each Δ, this sequence of subdeterminants is fixed, the first one being Δ itself, and the first subdeterminant which is nonzero determines the sign of $\Delta(\varepsilon)$. Thus, SoS can also be interpreted as a systematic way to break ties. Although, symbolic perturbation is under heated debate [16] and there are alternate schemes that produce theoretically shorter sequences of subdeterminants [8], the original SoS method of [5] works very well for 3D Delaunay triangulations. Here, we have two kinds of geometric primitives: an orientation test and the in-sphere test; and both tests can be expressed in terms of determinants.

Performance

He fact that we have to replace the evaluation of Δ by a sequence of subdeterminants of $\Delta(\varepsilon)$ is, in praxis, insignificant. Although, the sequence of subdeterminants for the full ε-polynomial can get rather long[4] the number of *actually* evaluated subdeterminants is low on the average for "typical" point sets. Moreover, the size of the subdeterminants decreases rapidly. Empirical tests with a variety of data, as reported in [14], show an average "evaluation depth" of less than 0.5, even for relatively degenerated point sets derived from volumetric data.[5]

The real "price" of SoS, however, is due to the fact that we have to use exact arithmetic; after all we have to test whether determinants are negative, zero, or positive, and we have to do so in an exact manner. We implemented a long-integer module which turns out to be considerably faster than any of the commonly available long-integer libraries, and, although, determinant evaluations contribute roughly 56% to the total number of CPU cycles (evaluation of geometric primitives overall accounts for 85%) [14], empirical run-time measurements are not at all discouraging:



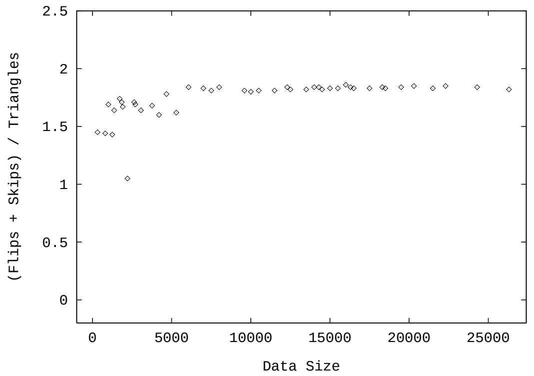
The experiments were run on SGI hardware, featuring 50 Mhz R4000 RISC processors, and always enough main memory to avoid page faults (e.g., for n in the 25k range,

[4]For $\varepsilon(i,j)$ chosen as in [5], the number of coefficients in $\Delta(\varepsilon)$ is exponential in d, the number of coordinates for each point; e.g., in 3D, the ε-polynomial for the in-sphere test has 50 coefficients.

[5]Let $\Delta = C_0, C_1, \ldots$ denote the subdeterminants of $\Delta(\varepsilon)$. If C_ℓ is the first one with $C_\ell \neq 0$, then we say this evaluation has a *depth* of ℓ.

we typically need 40Mb).

As mentioned earlier, the number of flips is roughly proportional to $n_f = |\mathcal{D}|$, the number of triangles in the final triangulation; but what are the constants? Experiments suggests that the algorithm typically needs less than $2n_f$ steps:



Recall that the algorithm is randomized. The above plots were generated by running each data set 13 times, and plotting the mean value. The confidence intervals are usually too small to be drawn; see [14].

Perturbation Artefacts

SoS does not allow to "selectively" remove certain degeneracies, while retaining others; it rather *fully* implements what theoreticians call "general position" on the input. Moreover, the resulting triangulation is for $S(\varepsilon)$ rather than S. For nondegenerate configurations in S, this does not matter: simply ignore the ε-perturbation and use the original coordinates; but for degeneracies in the data, some postprocessing might be desirable. Consider, for example, coplanar points on the boundary of the convex hull of S. The perturbation will move some of these points off the boundary, while some will remain on it. As a result, the points that were moved inside the convex hull, will be covered by "flat" tetrahedra in $\mathcal{D}$ of $S(\varepsilon)$. After ignoring the ε-terms of the coordinates, these tetrahedra will be total flat, so-called *volume-0 slivers*. Such slivers will occur quite frequently (typically, up to 25% of the tetrahedra can be volume-0 slivers!) and they might be awkward for the application. Fortunately, they can easily be removed, in a postprocessing step, using $O(1)$ time per sliver [14].

Another interesting "side effect" of the perturbation is that the resulting triangulation is guaranteed not to contain any volume-0 slivers inside the convex hull.[6] In other words, after the above postprocessing step, which removes the slivers on the convex hull, the resulting Delaunay triangulation $\mathcal{D}$ of S will not contain any volume-0 tetrahedra, no matter what input is given.

[6]This has to do with the fact that the perturbation is set up in such a way that $\varepsilon(i, j_1) \gg \varepsilon(i, j_2)$ for $j_1 < j_2$.

Input, Output, Data Structure

Detri 2.2 read ASCII input files, where each data line i contains the three coordinates of point i. The coordinates can be integers or in a user-defined fix-point format; internally, they will always be converted to long-integers. The output is a binary file, containing the *triangle-edge* data structure of the triangulation. This data structure is described in detail in [14]. It is a simplification of Dobkin and Laszlo's edge-facet structure [4], specialized for triangulations without duals. It uses 36 bytes per triangle, and fully represent the topology and order in a 3D triangulation. The user has now the option to either use the binary file of the data structure directly (the code for the access functions is provided), or, alternatively, use a run-time option in Detri to get an ASCII listing of the faces.

Availability

The `C` source code is available via anonymous `ftp` from `cs.uiuc.edu`, directory `pub/edels/geometry` (this directory also contains a collection of data sets, used in the experiments of [14]), or, as part of the 3D Alpha shape software, as it is available on `ftp.ncsa.uiuc.edu`, directory `Visualization/Alpha-shape/alpha-2.2`. The Detri distribution includes the above mentioned long-integer module, and the SoS implementation for a set of geometric primitives.

Acknowledgments

Version 2.2 of Detri was mainly developed while the author was completing his PhD under Herbert Edelsbrunner at the Department of Computer Science of the University of Illinois at Urbana-Champaign, where he was working on the 3D alpha shape project [6]. Obviously, the author is deeply indebted to Herbert for all his support. He also has to thank the National Center for Supercomputing Applications, especially, Ping Fu from the Software Development Group, for providing access to their computing facilities.

References

[1] F. Aurenhammer. Voronoi diagrams — a survey of a fundamental data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[2] A. Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166, 1981.

[3] B. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[4] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4(1):3–32, 1989.

[5] H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[6] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[7] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 43–52, 1992.

[8] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 74–82, 1992.

[9] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[10] L. J. Guibas and J. Stolfi. Primitives for manipulation of general subdivisions and the computation of Voronoi diagrams. *Transactions on Graphics*, 4(2):74–123, 1985.

[11] B. Joe. Three-dimensional triangulations from local transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(4):718–741, 1989.

[12] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8(2):123–142, 1991.

[13] C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, New York, 1977.

[14] E. P. Mücke. *Shapes and Implementations in Three-Dimensional Geometry.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Ubana, Illinois, 1993. Report UIUCDCS-R-93-1836.

[15] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proceedings of the 7th Annual Symposium on Computational Geometry*, pages 357–363, 1991.

[16] R. Seidel. The nature and meaning of perturbations in geometric computing. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science(STACS 94)*, volume 775 of *Lecture Notes in Computer Science*, pages 3–17, 1994.

[17] G. 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.

[18] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième Mémoire: Recherches sur les parallélloèdres primitifs. *Journal für die Reine und Angewandte Mathematik*, 134:198–287, 1908.