Computing Voronoi Diagrams of Line Segments Reliably and Efficiently

Martin Held*
Institut für Computerwissenschaften
Universität Salzburg
A-5020 Salzburg, Austria

Abstract

We study an engineering approach to computing Voronoi diagrams of points and line segments in the two-dimensional Euclidean space. Our Voronoi code, named vroni, uses standard floating-point arithmetic. It is based on Sugihara and Iri's topology-oriented approach, a very careful implementation of the numerical computations required, an automatic relaxation of epsilon thresholds, and on a multi-level recovery process combined with "desperate mode". Vroni was tested extensively on real-world data and turned out to be reliable. CPU-time statistics document that it is always faster than other popular Voronoi codes.

1 Introduction

In a recent editorial, Fortune [2] wrote that "it is notoriously difficult to obtain a practical implementation of an abstractly described geometric algorithm". According to the author's personal experience this remark is particularly true for the implementation of Voronoi diagrams of line segments. This paper discusses the design and implementation of a reliable and efficient Voronoi code, named vroni, that is able to cope with any form of polygonal input data, be it clean or not. Specifically,

- we *describe a topology-oriented implementation* based on the approach by Sugihara, Iri, and Imai [8, 5, 9];
- we *reveal missing details* of the algorithm description given by Imai [5];
- we discuss *reliability issues* related to the handling of data that is not clean;
- we explain how a *reliable and efficient* code that uses *conventional floating-point arithmetic* has been engineered; and
- we report on a *thorough experimental comparison* with other publicly available Voronoi codes.

To our knowledge, vroni is the first code for computing Voronoi diagrams of line segments which is both reliable and fast.

This paper is accompanied by several color plates available on the WWW. Point your browser to the WWW home-page of vroni:
www.cosy.sbg.ac.at/~held/projects/vroni/vroni.html.
This WWW page also provides a draft of the full version of this paper.

2 Survey of vroni

The success of our triangulation code FIST [3, 4] motivated us to apply the same design principles to the computation of Voronoi diagrams of point sites and line segments. Its basic idea is to compute the Voronoi diagram such that

1. the correctness of each step can easily be monitored,
2. the correctness of the overall algorithm is a simple consequence of the correctness of each step,
3. the "successful" completion of each step can be enforced in case of algorithmic or numerical problems.

Our code, vroni, computes the Voronoi diagram in a randomized incremental manner, based on the topology-oriented approach by Sugihara et al. [8, 5, 9]. Starting with an initially empty set of processed sites (and a corresponding trivial Voronoi diagram), the final Voronoi diagram is obtained by incrementally adding one new site to the set of processed sites and updating the Voronoi diagram accordingly. The incremental updates are carried out in a way such that the following

*Partially supported by grants and hardware donations from Sun Microsystems, and by NSF Grant CCR-9732220. Part of this work was carried out while the author visited SUNY Stony Brook. Email: held@cosy.sbg.ac.at; URL: http://www.cosy.sbg.ac.at/~held/held.html.

minimum set of topological requirements for a graph to form the Voronoi diagram (node/edge graph) of a set of sites are fulfilled ([9]):

- Every site has its own Voronoi region.
- Every Voronoi region is connected.
- Two Voronoi regions of point sites share at most one edge (Voronoi bisector).
- The Voronoi region of a segment is adjacent to the Voronoi regions of its endpoints.

Every update of the Voronoi diagram is performed by deleting old Voronoi nodes (and inserting new Voronoi nodes) according to these topological properties. Numerical predicates are used to select those nodes that are the best candidates for deletion.

Provided that every incremental update is performed successfully, we are bound to obtain a final Voronoi diagram which also exhibits the set of topological properties. The obvious crux is to guarantee that every incremental update is performed successfully. This task splits into three completely different subtasks:

1. Handling all purely algorithmic problems in order to preserve the topological properties when the algorithm is applied to clean data;
2. Extending the basic algorithm to data that need not be clean; and
3. Handling numerical inaccuracies due to the use of (inexact) floating-point arithmetic.

Subtask 1 merely means filling in the details missing in Imai's paper [5]. As explained in [5, 9], the set of nodes/edges to be deleted from the Voronoi diagram during an incremental update has to form a tree. However, we cannot simply stop deleting any node whose deletion would cause a node/edge cycle to be deleted: the insertion of a line segment may result in node/edge cycles for which all nodes are to be deleted indeed, but a portion of one edge has to be preserved. In order to solve this problem we scan all edges (bisectors) of a cycle and check whether they contain a point that is guaranteed to be closer to its defining sites than to the new line segment.

Subtask 2 involves the identification of data which is not clean, on-the-fly local cleaning of the data by inserting points of intersection and removing any duplicate (portions of) sites, and the amendments necessary in order to handle the resulting data during the incremental update. In particular, this requires the handling of "spikes", i.e., two or more line segments being incident at one endpoint. Here, the problem is that the Voronoi region of the common endpoint of a spike has zero area — all its Voronoi nodes coincide with this endpoint. Following Imai's standard algorithm [5], the insertion of one new line segment incident at this endpoint would cause the deletion of all those degenerate nodes and would thus destroy this Voronoi region. This problem is solved by marking and preserving all degenerate nodes during the incremental updates.

Subtask 3 is the only subtask which is specific to our goal of using conventional floating-point arithmetic. This goal is motivated by the need to achieve processing times which are acceptable in practice, and by our desire to avoid any potential problems associated with porting (and licensing) somebody else's arithmetic library. Also, the re-conversion from exact reals to standard floating-point output seems to be (too) sensitive.

The basic principle of a topology-oriented algorithm is to value topological correctness and consistency higher than numerical correctness. It is obvious, though, that the quality of the output of any algorithm depends substantially on the fidelity of the numerical computations used. Thus, whenever a new Voronoi node is computed, we subject its coordinates to a series of sanity checks that shall help to detect any numerical garbage.

Subtask 3 is handled by a careful implementation of the numerical computations required, and by an automatic selection of a suitable precision threshold, dubbed "relaxation of epsilon thresholds". Starting with maximum precision, as given by the machine precision, vroni automatically increases its precision threshold and repeats any computation whose output failed a topological or numerical check. The precision threshold is repeatedly increased until the computation succeeds or a maximum error (as specified by the user) is reached.

Common to all three subtasks is the need to ensure that every computational step will always yield a result which is an acceptable input for a subsequent step. Typically, problems (such as arithmetic operations which cannot be performed, like divisions by zero) will be due to numerical inaccuracies or due to data which is not clean. We employ a multi-level recovery process which starts with back-up procedures that attempt to compute the data sought in a different way, gradually relaxes the (local) numerical correctness of the Voronoi diagram under computation, and finally applies "desperate mode".

Basically, at the stage of desperate mode, we replace "correct" or "optimum" by "best possible" and neglect the numerical sanity checks, restricting our attention to the pure topology-oriented approach. (Desperate mode is a concept which we have successfully used in the triangulation code FIST, cf. [3, 4].) It is important to point out that vroni always first attempts to perform a computation as it would normally perform it, irrelevant of whether or not it has already entered desperate mode once before.

Anyway, even if the numerical sanity checks should fail continuously – e.g., due to an intentionally specified odd precision threshold – vroni will not get stuck, crash or loop. Rather, due to the topology-oriented approach combined with desperate mode it will still output a structure that meets the topological conditions specified. (We admit that its output may nevertheless be of little practical use in the event of such a worst-case scenario.)

Summarizing, the design and implementation of vroni is based on the following *four corner stones*:

1. Sugihara and Iri's *topology-oriented approach* [8, 9];

2. a very *careful implementation* of the numerical computations required;

3. an automatic *relaxation of epsilon thresholds*; and

4. a *multi-level recovery process* combined with *desperate mode.*

3 Experimental Results

A major goal of this work was to prove experimentally that this algorithm is both reliable and efficient. We tested the reliability of vroni on about 2,000 real-world polygonal data sets which we obtained from companies and colleagues. The complexity of our real-world test data ranged from a few tens of segments to more than 450,000 segments, with most data sets having several thousand segments.

In order to evaluate the performance of vroni we opted for a different approach, based on synthetic test data. We generated 10 polygons each for sets of 2^i vertices, where $6 \leq i \leq 15$. (Thus, the numbers of segments ranged between 64 and 32,768.) This procedure was repeated for four different classes of test polygons that reflect different types of polygons, ranging from fairly smooth polygons with short edges to polygons that exhibit a large number of long edges and sharp corners. All synthetic polygons were generated by means of RPG, cf. [1].

The cpu-time consumption of our code, and of the other codes, was obtained by using the C system function "getrusage()". We report both the system and the user time. Of course, any file I/O and similar preprocessing is not included in the timings reported. All cpu times are given in milliseconds.

Besides our own code, vroni, we tested the following codes:

- Imai's Fortran code plvor [5],
- Seel's C++ code avd [6],
- Sethia's C++ code pvd [7].

All tests reported here were carried out on a Sun Ultra 30, running Solaris 2.6 on a 296MHz processor. (Our test machine had 384MB of main memory, but memory clearly was no issue for any of the codes.) Imai's plvor (version 0.9), Sethia's pvd and vroni were compiled with the SunPro cc/CC/f77 compiler, using the optimization level "-xO4". Seel's avd (version 1.1, based on LEDA 4.1) was compiled with GNU's g++, using the optimization level "-O". (Our attempts to compile it with the SunPro CC failed.)

Imai's plvor repeatedly generated cryptic warning messages and ran into floating-point exceptions for collections of line segments that did not intersect except at their endpoints. A visual inspection of some of the Voronoi diagrams generated revealed that those warning messages were an indication of serious deficiencies: if plvor had output multiple warning messages then the numerical properties of the Voronoi diagram generated were seriously flawed, with Voronoi edges scattered around in no apparent relation to the input data. Thus, plvor cannot handle a good deal of our test data.

Seel's avd performed correctly on all data sets that we applied it to. It seems to handle any arrangement of point sites and line segments successfully as long as the data is not corrupted. (It dies gracefully when it encounters two line segments that intersect in an interior point). As documented below, it is too slow to be applied to large data sets, and we ran it only on data sets with 2,048 or less points/segments. Thus, our tests of avd are not conclusive. However, it seems that a user is more likely to loose his or her patience than to see avd fail.

Sethia's pvd is restricted to clean polygonal data that forms the boundary of a multiply-connected area. Thus, it is not applicable to most of our real-world test data. It seems to handle clean data fairly reliably. (We saw it crash occasionally, though.)

Representative average timings for our synthetic polygons are summarized in Table 1. (Entries marked by "n/a" are not available due to an excessive cpu-time consumption.) We realized soon that vroni is the only code that is not affected by the geometric distribution of the input segments, while all three other codes get noticeable slower for polygons that exhibit some form of "degeneracy", such as smooth approximations of curved objects, a large number of sites being cocircular (such as straight-line approximations of circular arcs), or large numbers of individual polygons. See Table 2 for a listing of test results for increasingly finer polygonal approximations of an ellipse.

Of course, the reader is cautioned that any comparison of algorithms implemented by different programmers and compiled by different compilers has to be taken with a grain of salt. Our experience tells us that a

size	synthetic			
	avd	plvor	pvd	vroni
64	50.55	1.359	0.164	0.090
128	75.26	0.922	0.160	0.117
256	119.8	0.657	0.162	0.112
512	209.6	0.565	0.158	0.117
1024	404.4	0.517	0.167	0.121
2048	758.9	0.511	0.167	0.123
4096	n/a	0.561	0.175	0.131
8192	n/a	0.643	0.185	0.141
16384	n/a	0.898	0.209	0.151
32768	n/a	1.293	0.234	0.161

Table 1: Average cpu times (per segment) in milliseconds for the different Voronoi codes applied to the "synthetic" polygons.

size	ellipse			
	avd	plvor	pvd	vroni
64	82.66	1.406	0.156	0.156
128	133.4	1.016	0.312	0.078
256	262.9	1.211	0.469	0.078
512	674.2	3.105	0.664	0.117
1024	1892.	11.08	1.182	0.107
2048	6059.	43.38	2.222	0.107
4096	n/a	49.69	4.187	0.117
8192	n/a	52.34	8.179	0.122
16384	n/a	146.9	18.61	0.133
32768	n/a	647.4	38.56	0.143

Table 2: CPU time (per segment) for the different Voronoi codes applied to the "ellipse" polygons.

different compiler/optimization set-up should indeed be expected to yield slightly different test results, but any cpu-time comparison among these four Voronoi codes can expect to see vroni as the winner.

4 Conclusion

We discussed an incremental algorithm for computing Voronoi diagrams of points and line segments in 2D. Extensive experimental tests with both synthetic and real-world test data have shown that the resulting code, vroni, is reliable.

An experimental comparison demonstrated clearly that vroni is faster than other popular Voronoi codes. In particular, it is orders of magnitude faster than avd, which is the only (publicly available) Voronoi code that could also handle all the data we applied it to.

Work on extending vroni to circular arcs has been started. Future work will also include the practical coupling of vroni with a library for doing exact arithmetic.

Acknowledgments

This work has benefited from discussions with Joe Mitchell and Saurabh Sethia (SUNY Stony Brook). Our experiments were run on a Sun Ultra 30 donated by Sun Microsystems, and involved Voronoi codes made publicly available by Michael Seel and Toshiyuki Imai. We could test our code on real-world data obtained from Mike Ady, Gill Barequet, Jack Snoeyink, Markus Thoma, Baltic Drainage Basin Project, Bridgeport Machines, British Antartic Survey, Digital Chart of the World, EOS GmbH, Minnesota River Basin Data Center, and Taglio s.r.l.

References

[1] T. Auer and M. Held. Heuristics for the Generation of Random Polygons. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 38–44, Ottawa, Canada, Aug 1996. Carleton University Press.

[2] S. Fortune. Introduction. *Algorithmica*, 27(1):1–4, 2000.

[3] M. Held. Efficient and Reliable Triangulation of Polygons. In *Proc. Comput. Graphics Internat. '98*, pages 633–643, Hannover, Germany, June 1998.

[4] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 2000. To appear.

[5] T. Imai. A Topology Oriented Algorithm for the Voronoi Diagram of Polygons. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 107–112, Ottawa, Canada, Aug 1996. Carleton University Press.

[6] M. Seel. An Accurate Arithmetic Implementation of Line Segment AVDs. Technical report, MPI-Saarbrücken, D-66123 Saarbrücken, Germany, Sep 1996.

[7] S. Sethia, J.S.B. Mitchell, and M. Held. PVD. Dept. of Appl. Math. Stat., SUNY at Stony Brook, NY, USA, 1999.

[8] K. Sugihara and M. Iri. Construction of the Voronoi Diagram for 'One Million' Generators in Single-Precision Arithmetic. *Proc. of the IEEE*, 80(9):1471–1484, Sep 1992.

[9] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-Oriented Implementation - An Approach to Robust Geometric Algorithms. *Algorithmica*, 27(1):5–20, 2000.