Time, space, and precision: Revisiting classic problems in computational geometry with degree-driven analysis

Jack Snoeyink*

Computational geometry, as a branch of the theory of computer science, designs and analyzes data structures and algorithms most often for a RealRAM model, which has three unbounded quantities: the time its program can run, the number of memory cells, and the number of bits that can be stored in each cell. Since actual computers have constraints for time and memory — constraints that change as technology advances — we follow the computer science tradition of developing our algorithms to minimize time and memory size, measured using the familiar asymptotic, big-O notation, and tacitly agreeing not to exploit the third unbounded quantity in our model. (Exceptions published with capitalized words in the title [3] merely prove the rule.)

Actual computers, however, do not faithfully implement the abstractions of an infinite real number line or Euclidean plane. For example, the floating point coordinates calculated as the intersection of two line segments may lie on neither line!

For many numerical calculations, we are happy with approximate answers—our input may be accurate to only five decimal places, so the best we may hope for is an answer within $\pm 10^{-5}$. But geometric tests usually ask for the exact sign of a calculation on coordinate values—the closer an answer is to zero, the more bits of accuracy are needed. Because efficient geometric algorithms infer the results of many tests from the few tests that they perform, an incorrect test can cause catastrophic failure in rare, and therefore almost impossible to debug, cases.

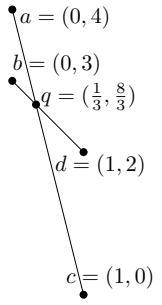

Figure 1: In floating point, q is below $\overline{ac}$, $\overline{bd}$.

The most common way to address these problems is to build geometric algorithms on top of some library that more faithfully implements the abstraction of computation on real numbers, as done in LEDA, CGAL, and Core, and by numerous research papers on floating point filters, interval and affine arithmetics, and lazy evaluation [2, 5].

This talk is about a less common way: by including into the machine model some notion of the precision required by a computation (in the form of the degree of a polynomial whose sign is being evaluated) it allows the algorithm designer to consider precision as a resource, just as we traditionally consider memory and running time as resources and try to find algorithms that minimize their use. This Degree-Driven Algorithm Design (DDAD) was proposed in a 1999 paper of Liotta, Preparata, and Tamassia [4]. My this talk summarizes NSF-funded work on their program with several students, including David Millman, Andrea Mantler, Clinton Freeman, Sajal Dash and Vince Miller.

I'll give examples of the algorithm development process under this model for map overlay or polygon boolean operations via a simpler implementation of red/blue line segment intersection that takes n line segments and computes their k intersections in order along each segment using $\Theta(n \log n + k)$ time and double precision. The classic Bentley-Ottmann algorithm [1] takes $\Theta((n + k) \log n)$ and five-fold precision. I'll also give some other examples of favorites: digital hulls, distance transforms, and Voronoi diagrams, demonstrating how incorporating precision as an additional constraint or optimization target gives us a chance to re-visit standard problems, and inspires new solutions, and carries proof down to the implementation level.

References

[1] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, 1979.

[2] C. Burnikel, J. Könemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Exact geometric computation in LEDA. In *Proc. 11th SoCG*, pages 418–419. 1995.

[3] M. L. Fredman and D. E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. In *STOC*, pages 1–7, 1990.

[4] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: an illustration of degree-driven algorithm design. *SIAM J. Computing*, 28(3):864–889, 1999.

[5] S. Pion and A. Fabri. A generic lazy evaluation scheme for exact geometric computations. *Sci. Comput. Program.*, 76(4):307–323, Apr. 2011.

*Department of Computer Science, UNC Chapel Hill, email: snoeyink@cs.unc.edu. Supported by NSF grant 1018498.