Robustness issues in geometric algorithms

Steven Fortune
Bell Laboratories
Murray Hill, NJ

March 1, 1996

Geometric algorithms are usually described assuming that input data is in general position and that exact real arithmetic provides reliable geometric primitives. Often an implementer substitutes floating-point arithmetic for real arithmetic and uses real-world data, which might be degenerate by accident or design. Hence, the correctness proof of the mathematical algorithm does not extend to the program, and the program can fail on seemingly appropriate input data. This is the well-known problem of "robustness" in geometric computing[11].

This note[1] surveys recent research and current trends on the issues of numerical precision and degeneracies, addressing the assumptions of exact real arithmetic and general position, respectively. The references are impressionistic rather than complete, with the emphasis on recent rather than older work. Other surveys of robustness issues are also available [6, 11, 15].

1 Numerical precision

A geometric predicate is determined by the sign of an arithmetic expression. Sign-evaluation is exact in the conceptual model of the real numbers. However, a computer implementation cannot use real numbers. There are two scientifically plausible substitutions for exact real arithmetic. One is to substitute floating-point arithmetic for real arithmetic, and somehow deal with the resulting rounding error. The other is to substitute exact arithmetic on a subset of the reals, such as the integers or the rationals.

Floating-point arithmetic. Floating-point arithmetic is the default choice for much scientific programming. Used naively, it does not provide reliable geometric predicates. If an instance of a predicate is nearly degenerate, then the value of the corresponding expression can be very small, less than the rounding error in the floating-point evaluation of the expression. Hence the sign of the computed expression may well be erroneous. Usually it is possible to argue that the computed expression sign is the true sign for slightly perturbed coordinate data. Since coordinate data may well be imprecise originally, the erroneous sign may appear to be innocuous. The difficulty arises with multiple predicate evaluations; there is no guarantee that any single global perturbation produces all the computed predicate values. Indeed, the computed predicate values may be geometrically inconsistent. Catastrophic implementation failure can easily result.

There are two broad categories of methods to deal consistently with floating-point rounding error. One category formalizes the notion of tolerances [5, 12]. A typical strategy might associate inner and outer tolerances with an object, say a point. If the inner tolerances of two points intersect, they are deemed coincident and merged; if the outer tolerances are disjoint, they are deemed separate; if neither case holds, the situation is ambiguous. As long as no ambiguity results during a computation, the result is correct. The hope is that ambiguities arise infrequently; an obvious drawback of this strategy is that is not clear what to do if an ambiguity does arise. Another drawback is that the generalization to complex geometric objects is not straightforward.

A second floating-point approach is modeled on the error analysis of numerical methods, particularly linear algebra. The goal is to show that a suitably implemented algorithm provides an answer that is in some precise sense near the mathematically correct answer. Error analysis of geometric algorithms requires consideration of both combinatorial and numeric structure. Often it is easy to argue that an algorithm produces combinatorially valid output[18], at least with suitably relaxed requirements. It has turned out to be much more difficult to argue that the numeric error associated with combinatorial structure is small. Full error analysis has been carried only for a few simple algorithms [8, 15].

[1] This note was adapted from a chapter of the report "Computational geometry: perspectives and challenges", edited by Bernard Chazelle.

Exact computation. Exact geometric computation requires that every predicate evaluation be correct [19]. This can be achieved either by computing every numeric value exactly, or by some symbolic or implicit numeric representation that allows predicate values to be computed exactly. Exact computation is theoretically possible whenever all the numeric values are algebraic, which is the case for most current problems in computational geometry.

With exact geometric computation, it is no longer reasonable to assume that each arithmetic operation takes constant time, as is the case with floating-point arithmetic. Rather, the cost of an arithmetic operation depends upon its context. Simple geometric predicates can be expressed as the sign-evaluation of an integer polynomial. The required arithmetic bit-length can be estimated from the degree of the polynomial and the bit-length of coordinates. For many predicates involving linear objects (such as orientation predicates) the degree is small, and the required bit-length is relatively minimal. However, even in simple cases software arithmetic is required, say over the integers or rationals, with a resulting increase in performance cost.

The cost of arithmetic also increases because of geometric constructions. A geometric constructor produces a new geometric object from old objects. Typically the coordinates of the new object can be expressed as polynomials in the coordinates of the old objects, and hence bit-length can be estimated from polynomial degree. For example, the coordinates of the intersection point of three planes have bit-length about three times the plane coefficient bit-length; the coefficients of a plane through three such points have bit-length about nine times the original plane coefficient bit-lengths. Hence an algorithm that uses geometric objects constructed to arbitrary depth can require arithmetic with prodigious bit-length, even if the algorithmic predicates are relatively simple.

Complex primitives on linear objects, or simple primitives on curved objects, apparently require that arithmetic be augmented with square roots, and, more generally, arbitrary polynomial roots. Such arithmetic operations can be implemented symbolically using general algebraic techniques such as resultants, Sturm sequences, and root separation bounds.

Discussion. Exact arithmetic is the most general technique that can guarantee the numerical reliablity of a geometric implementation. It appears not to be used widely, perhaps because of perceived performance cost, unfamiliarity, or lack of appropriate software tools.

There has been considerable recent progress in reducing the performance cost of exact arithmetic, at least for primitives with relatively minimal bit-length requirements. A useful strategy is adaptive-precision arithmetic. The predicate expression is first evaluated with low-precision arithmetic, perhaps with floating point, and only reevaluated at higher precision if the uncertainty bound exceeds the magnitude of the computed value. This strategy admits a rich spectrum of implementations [7, 13, 14, 17] with further exploration possible. Alternatively, there has been some progress at reducing the bit-length required for evaluation of simple predicates [3].

High-level geometric rounding algorithms may be able to address the coordinate growth that results from geometric constructions. Consider a polyhedron in three dimensions, with vertices, edges, and faces specified both by incidence structure and by coordinates. An exact computation may result in coordinates with large bit-length. An application may be satisfied with a short bit-length approximation to the numeric coordinates, as long as the combinatorial and numeric data are consistent. Rounding the coordinates of a polyhedron is not straightforward, however, since its combinatorial structure may be invalidated by small perturbations of its faces or vertices. A few examples [9, 10] demonstrate the complexity that results when high-level rounding is considered an integral part of a geometric algorithm; much more research needs to be done.

Efficient exact computation of geometric predicates involving algebraic numbers is just beginning to be explored [1, 14, 16, 19]. Many predicates that are well-understood in the linear domain (e.g. orientation) become much more complex in the curved domain, and their best evaluation using exact arithmetic is not well understood. Classical algebraic techniques can be used to give bit-length estimates required for expression evaluation; in many cases the bounds are dauntingly large [20]. It is possible that such bit-length estimates are too pessimistic, either in theory because the underlying algebraic machinery is not developed enough to give sharp estimates, or in practice because instances requiring long bit-length are infrequent.

2 Degeneracies.

Degeneracies arise from the special position of two geometric objects. For example, two segments in general position either do not intersect or intersect at a point interior to both segments. Two intersecting segments in special position may overlap, may share a common endpoint with or without being collinear, may have one segment endpoint interior to the other segment, etc. Real-world data is likely to be degenerate. For example, segment endpoints may be explicitly chosen from a coarse grid, to facilitate interactive design.

The effect of degeneracy is to increase the number of special cases. While a sorting algorithm must deal only with the possibility of two keys being equal, a typical geometric algorithm faces the possibility of dozens or hundreds of different special cases. Since the overall correctness of an implementation may depend upon the correct treatment of special cases, the handling of special cases can permeate the implementation. This raises the obvious reliability concern that all cases have been considered and correctly handled.

Symbolic perturbation schemes [4] allow degeneracies to be resolved automatically. Conceptually, each geometric coordinate c_i is replaced with a symbolically perturbed coordinate $c_i + f_i(\epsilon)$, where $\epsilon > 0$ is unknown but very small and the perturbation function f_i is simple, say a polynomial. Substitution of the symbolically perturbed coordinates in a predicate expression results in a polynomial in ϵ with coefficients determined by the original geometric coordinates. The sign of the expression is given by the sign of the first nonzero coefficient, with coefficients taken in order of increasing powers of ϵ. For many classes of predicates, the f_i can be chosen to resolve all degeneracies.

Discussion. Symbolic perturbation is the subject of some controversy [2]. While it is certainly a useful tool in the implementation of geometric algorithms, existing schemes are not as applicable as might be desired. First, symbolic perturbation requires exact arithmetic, since the correctness of the perturbation depends upon exact evaluation of arithmetic expressions. Second, symbolic perturbation has been worked out in detail for only a small class of predicates. For example, constructed objects are often disallowed, since the perturbation function for a constructed object depends upon how the object was constructed and is much more complicated than the perturbation function for a primitive object. Perhaps most fundamentally, in a degenerate situation an algorithm implemented using symbolic perturbation does not solve the problem instance, but an arbitrarily-chosen nearby problem instance. This might be inappropriate in some applications, e.g, the symbolically-perturbed union of two polyhedra with abutting faces may not even be connected, if the perturbation pushes the faces apart.

3 Conclusions

Robustness has received considerable attention as a research area in the last few years, and many useful techniques have emerged. A sophisticated programmer can use the techniques to obtain an efficient implementation of a geometric algorithm that is as reliable as the implementation of a purely combinatorial algorithm.

However, the field is not mature, and the the application of the techniques is by no means routine. For example, a large application may well have both a kernel that requires complete reliability and outer layers that are less demanding. Thus the boolean operations of a solid modeler might require exact arithmetic to guarantee topological consistency, while the graphical rendering of a resulting solid can be done with less precision. Isolating a kernel that can be implemented robustly may be a significant design challenge.

References

[1] C. Burnikel, K. Mehlhorn, S. Schirra, How to compute the Voronoi diagram of line segments: theoretical and experimental results. *Proc. 2nd Eur. Symp. Alg. (ESA 94)*, 1994.

[2] C. Burnikel, K. Mehlhorn, S. Schirra, On degeneracy in geometric computations, *Proc. Fifth Annual Symp. Discrete Algorithms* pp. 16–23, 1994.

[3] K. L. Clarkson, Safe and effective determinant evaluation, *33th Symp. on Found. Comp. Sci.* 387–395, 1992.

[4] H. Edelsbrunner, E. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics* **9**(1):66-104, 1990.

[5] S. Fang, B. Bruderlin, X. Zhu, Robustness in solid modelling – a tolerance based, intuitionistic approach, *Computer Aided Design*, **25**:9, 1993.

[6] S. Fortune, Progress in computational geometry, in *Directions in Geometric Computing*, Ch. 3, pp. 81–128, R. Martin, ed. Information Geometers Ltd, 1993.

[7] S. Fortune, C. Van Wyk, Static analysis yields efficient exact integer arithmetic for computational geometry, to appear, *Transactions on Graphics*. See also Efficient exact arithmetic for computational geometry, *Proc. Ninth Ann. Symp. Comp. Geom*, pp. 163–172, 1993.

[8] S. Fortune, Numerical stability of algorithms for 2d Delaunay triangulations, *International Journal of Computational Geometry and Applications*, **5**(1,2), 193–213, 1995.

[9] S. Fortune, Polyhedral modelling with exact arithmetic, *Proc. Third Symp. Solid Modeling and Applications*, pp. 225-234, 1995.

[10] L. Guibas, D. Marimont, Rounding arrangements dynamically, manuscript, 1995.

[11] C. Hoffmann, The problems of accuracy and robustness in geometric computation. *Computer* **22**:31–42 (1989).

[12] D.J. Jackson, Boundary representation modelling with local tolerances, *Proc. Third Symp. on Solid Modeling and Applications*, pp. 247–254 (1995).

[13] P. Jaillon, Proposition d'une arithmétique rationnelle paresseuse et d'un outil d'aide à la saisie d'objets en synthèse d'images, Thèse, Ecole Nationale Superieure des Mines de Saint-Etienne, 1993.

[14] S. Näher, The LEDA user manual, Version 3.1, January 16, 1995. LEDA is available by anonymous FTP from `ftp.mpi-sb.mpg.de` in directory `/pub/LEDA`.

[15] Victor Milenkovic, Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, **37**:377–401, 1988.

[16] A. Rege, J. Canny, Fast point location for two- and three-dimesional real algebraic geometry, to appear, 1995.

[17] J. R. Shewchuk, Adaptive precision floating-point arithmetic and fast robust geometric predicates in C, manuscript, Carnegie-Mellon University, 1995.

[18] K. Sugihara, M. Iri, Construction of the Voronoi diagram for one million generators in single precision arithmetic, *First Can. Conf. Comp. Geom.*, 1989.

[19] C. Yap, T. Dubé, The exact computation paradigm, 452–492, *Computing in Euclidean geometry*, D.Z. Du, F. Hwang, eds, World Scientific, 1995, second edition.

[20] J. Yu, Exact arithmetic solid modeling, Ph.D. Thesis, Purdue University, 1992, available as CSD-TR-92-037.