数理解析研究所講究録
第1641巻 2009年 99-105

Unsolved Problems in Robustness of Geometric Algorithms

Kokichi Sugihara
Department of Mathematical Informatics
Graduate School of Information Science and Technology
The University of Tokyo
e-mail: sugihara@mist.i.u-tokyo.ac.jp

Abstract

Geometric algorithms are not necessarily robust against numerical errors, because their correctness is guaranteed only in the error-free ideal world. Many approaches have been proposed to attain robustness, but still many problems remain unsolved. We first review existing approaches, next list major unsolved problems, and finally discuss possible directions to solve those problems.

1 Introduction

Robustness is one of the most serious issues in applications of geometric algorithms to practical problems. Algorithms, even though their correctness has been proved mathematically, do not necessarily have a stable behavior when they are implemented in computers. This is mainly because there is a great difference between the theoretical world where the correctness of the algorithms is proved and the practical world where the algorithms are implemented and used. Indeed, the theoretical world is constructed on the assumption that given information is perfect and computation can be done without errors, whereas the practical world is contaminated with many errors, uncertainties and noises. Therefore, a theoretically correct algorithm can easily fail when executed in a computer.

The importance and seriousness of this issue have long been recognized, and many approaches have been proposed to overcome the difficulty. Hence, it seems to be widely believed that robustness could be achieved by these approaches, and consequently theoreticians should concentrate on the mathematical correctness of the algorithms they design. However, the current situation is far from complete; many problems remain to be overcome to attain numerical robustness in the real world.

This paper first reviews major existing approaches to the robustness of geometric algorithms, next shows remaining problems, and finally discusses possible directions to solve these problems.

2 Existing Approaches and Their Limitations

Many methods have been proposed for designing numerically robust geometric algorithms. These can be classified into three approaches according to what extent they rely on numerical values. They are: the tolerance approach, the exact-computation approach, and the independent-test approach. In this section, we briefly review these approaches.

2.1 Tolerance Approach

In the first approach, numerical values are relied on moderately. Numerical errors generated in computers are usually very small. Therefore, we can expect inconsistencies caused by numerical errors to be rare, and we can treat them as exceptions. This idea can be categorized as the "tolerance approach". In this approach, every numerical computation for a geometric test is accompanied by its error analysis, and according to the estimated error bound, the result of the test is judged as "yes", "no", or "unknown". For the "yes" and "no" cases, they follow the procedure specified by the theoretical algorithm, whereas for the "unknown" case, they follow the exceptional branches of processing. Typical examples of the methods belonging to these approaches are epsilon geometry [1] and the hidden-variable method [3].

However, the way of making the exceptional branches usually depends on individual problems and individual cases, and is far from general. Moreover, the resulting procedure becomes complicated, because every geometric test generates three branches of processing (i.e., the "yes", "no", and "unknown" cases) instead of two. Furthermore, it is not easy to guarantee the correctness of the resulting three-branch procedure. Hence, this approach was successfully applied only to a limited area of simple problems.

2.2 Exact Computation Approach

In the second approach, numerical values are treated as perfectly reliable information. The simplest way to avoid inconsistency is to avoid having numerical errors at all, and this can be done by employing exact arithmetic, if possible. Suppose, for example, that the input data are strictly correct rational numbers and the algorithm employs only the four basic operations $(+, -, \times, \div)$. Then, the results of computation can be precisely represented by rational numbers. They contain no errors, and hence no misjudgments, resulting in robust behavior of the algorithms. This approach to robustness is called the "exact arithmetic approach".

The exact arithmetic approach is too restrictive, because it cannot handle irrational numbers. However, this restriction can be removed, because what we want to avoid are misjudgments about the topological structure. Topological structures are usually judged by the signs of computed numbers, and hence we can guarantee consistency if the judgments are always done correctly, even if the computed numbers themselves contain errors. Therefore, we attain robustness if we can provide multiple, but still finite, precision that is precise enough to recognize the signs correctly [7]. This approach is called exact

geometric computation. This name was coined by Yap [9].

Let us collect the exact arithmetic approach, the exact geometric computation approach, and similar in one group, and name it as the "exact computation paradigm". This paradigm is simple and powerful in the sense that consistency is automatically guaranteed by correct recognition of topological structures. For this simplicity, we have to pay an expensive cost for multiple, and hence slow, computations. Thus, the main concern in this paradigm is how to make the computation faster.

The most common method for acceleration is the floating-point filter. In this method, we first try to decide the sign of a computed value by floating-point arithmetic, and if the result is unreliable, we switch to exact arithmetic. Other acceleration methods include the sophisticated use of floating-point operations for correct judgment of the sign of the sum of real numbers and of an inner product.

In the exact computation paradigm, degeneracy is detected exactly, because the sign of a computed value can become exactly 0. Thus, the algorithm is not complete unless all degenerate cases are handled. To avoid this complication, we can use the symbolic perturbation technique, by which degeneracy is automatically avoided by adding infinitesimally small terms to input data.

This paradigm is used in many software libraries such as LEDA [2], CGAL and EGC Core Library [9].

2.3 Topology-Oriented Approach

The third approach is quite opposite to the exact computation approach, in the sense that it does not rely on the correctness of numerical values at all. It is important to note that numerical errors do not directly cause algorithms to fail. Numerical errors sometimes cause misjudgments of the combinatorial/topological structure of geometric objects; the misjudgments sometimes generate inconsistencies and the inconsistencies cause algorithms to fail. Hence, what we have to avoid to achieve robustness is inconsistency, but not necessarily numerical errors.

The third approach, called the "topology-oriented approach", is based on this recognition. In this approach, topological tests done in the algorithm are divided into two groups; the first group is a maximal set of "mutually independent" tests, and the second is the group of the remaining tests, where the mutually independent tests mean that the result of one test, i.e., the truth value, does not affect the result of the others.

We use numerical computation to evaluate the tests in the first group. Numerical errors may generate incorrect results of the tests. However, since they are independent, the results of the tests do not contradict each other; they form a consistent world.

For the other group of tests, instead of using numerical computation, we adopt the logical consequences of the results of the tests of the first group. Hence, the resulting values of all the tests are consistent.

Let us name this approach the "independent test paradigm".

An algorithm designed in this paradigm has many good characteristics [8]: (i) it runs fast because floating-point arithmetic can be used, (ii) it will never fail because

the consistency is guaranteed, no matter how low is the arithmetic precision, (iii) it always gives some output and the output is topologically consistent, and (iv) degeneracy is automatically avoided because inexact arithmetic cannot recognize degeneracy at all.

3 Remaining Problems

The two paradigms, the exact computation paradigm and the independent test paradigm, are powerful and give completely robust geometric algorithms whenever they can be successfully applied. However, they are not omnipotent; there still remain many difficulties. Major difficulties are as follows.

3.1 Incorrect Input

Input to an algorithm is not necessarily correct. For example, suppose that a polyhedron is given as the input. The description of the polyhedron is composed of topological data specifying the incidence relations among vertices, edges and faces, and the metric data such as the coordinates of the vertices. Numerical data are usually represented by rounded values and hence are not exactly correct. On the other hand, the exact computation paradigm is based on the assumption that the input is correct, and hence tries to treat the input data as exactly correct information.

However, numerical data may contradict topological data. Suppose that the topological data say that a face contains four vertices. Then, the exact computation paradigm expects these four vertices to be numerically coplanar. However, the coordinates of the vertices are represented inexactly, and hence the four vertices may not be coplanar. Similarly, even if the topological data state that four faces are incident to a common vertex, they may not be concurrent in the numerical sense. In that case, the input is inconsistent, and hence cannot be used in the exact computation paradigm.

3.2 Cascaded Processing

Results of computation require higher precision if we want to represent them precisely. Hence, if we apply computation repeatedly, the required precision raises quickly in an exponential order. This happens, for example, if we iteratively apply set-theoretic operations to polyhedra. In the exact computation approach, this will result in explosion of the required precision. In the topology-oriented approach, this will result in amplification of geometric disturbance.

Therefore, one problem here is how to minimize the growth of arithmetic precision or the growth of geometric disturbance. Sometimes the choice of the basic numerical data will help. For example, in the Boolean combination of polyhedrons, explosion in precision can be avoided if we choose the coefficients of the face equation as the basic numerical data instead of the coordinates of the vertices [7]. However, this kind of individual convention cannot be a general-purpose method.

3.3 High-degree Objects

Linear objects, such as lines and planes, require only a moderately high precision for exact computation. However, nonlinear objects, such as curved surfaces, require much higher precision for exact representation. Therefore, sometimes exact manipulation of these objects demands intractably high precision.

The topology-oriented approach can be formally applied to these situations, because floating-point arithmetic is used. However, it likewise encounters difficulty because the rounding errors are relatively large and they will cause (consistent, but) incorrect topology, which may generate large geometric disturbance.

3.4 Unwanted Side Effect of Perturbation

Symbolic perturbation is a powerful technique to avoid exceptional branches of processing in exact computation, because it completely removes all the degeneracy. However, sometimes we want to leave part of the degeneracy as it is, because otherwise an unwanted side effect takes place. For example, the Delaunay tetrahedrization is a useful tool for generating meshes for the finite element method. However, if we apply symbolic perturbation, four vertices of a square may generate a zero-volume tetrahedron [6], which should be avoided in practical applications such as finite element methods and interpolations. Contact detection problems and packing problems are other examples in which we want to leave some part of degeneracy as is. Therefore, a selective perturbation scheme is required.

3.5 Ill-definedness of Real Problems

The robustness issue is a matter of practical applications of geometric algorithms. In this sense, we should not ignore another serious difficulty; that is, the gap between practical and theoretical problems. Practical problems arising in the real world are ill-defined in many cases, while the algorithms designed in computational geometry are mainly intended for well-defined problems. Hence, we always risk having to change the problems, although they originally come from the practical world, into practically meaningless ones unintentionally when we try to formulate them in a mathematical manner.

For example, suppose that we want to construct an algorithm for interpreting line drawings as polyhedral objects, as humans can do. The problem might be described mathematically as the problem of judging whether there exists a polyhedron whose projection coincides with the given line drawing. However, the result of this kind of mathematical formulation is too strict when compared with the flexible human behavior of object recognition. This can be understood if we consider a picture of a truncated pyramid seen from above; the correct projection is always degenerate in that the three side edges meet at a common point (corresponding to the apex of the pyramid), and hence numerical perturbation of the vertex positions will make the picture mathematically incorrect. Thus, the machine perception becomes different from human perception [4].

4 Directions for the Future

It seems that all the five major difficulties listed in the last section are really difficult for the exact computation paradigm. In this paradigm, the necessary precision is determined by the geometric problem, and consequently the main concern is to raise the speed of the arithmetic to the required precision. For this purpose we might use software techniques, existing hardware, or both, and might also be able to design new hardware. In these directions, we may be able to attack the second and the third difficulties, but some new ideas appear to be necessary in order to attack the first, the fourth and the fifth difficulties.

On the other hand, in the independent test paradigm, computation can be done in floating-point arithmetic and hence quickly; the main concern is to raise the quality of the result of the computation. From this point of view, only the fourth difficulty is serious, whereas the others might be tractable. Let us see how the independent test paradigm might be able to cope with these difficulties.

The inconsistency of the input comes from the assumption that all the information of the input is important. In other words, if we select only independent information from this, we will get consistent information. One such trial is a "resolvable sequence" [5], in which we select only an independent subset of data for unique determination of the polyhedron.

The rapid increase in the required precision in cascaded operations or in high-degree objects is serious to the exact computation paradigm, but is not serious to the independent test paradigm. As the independent test paradigm uses only floating-point arithmetic, the required precision does not increase, even if we repeatedly apply operations. The only concern is how to keep the quality of the resulting object shape, so we need to develop some techniques to select the optimal independent set or to collect the most probable information from redundant data and/or to select the optimal order of applications of the repeated operations.

The degeneracy issue is serious in both paradigms. It seems that this difficulty is deeper than other difficulties, because it comes from the nature of the geometric problems, but not the way of processing them. Consequently, it seems that we need case-by-case care for individual problems rather than a unifying general method.

The last difficulty, the ill-definedness of real problems, is also serious if we want to solve them using algorithms in computational geometry. Of course, the validity of the theory is based on the fact that they treat well-defined problems. We should not change this basic attitude, but it might be possible to attack ill-defined problems by the combinations of well-defined algorithms. At least we could succeed in attaining human-like flexibility in picture interpretation by removing redundant information from the algebraic structure of a line drawing using combinatorial theory; refer to [4] for the details.

5 Concluding Remarks

We have surveyed major difficulties that remain to be overcome for robust geometric computation. The robustness issue is related to practical applications in its nature, and

consequently the difficulties contain ill-definedness of real problems. One might think that ill-defined problems are outside the scope of computational geometry, but I believe that it is a fruitful direction of future studies in computational geometry to pay attention to this kind of gap between theory and practice. In addition, I feel that the "independent test paradigm" is one of the more promising approaches to these problems.

Acknowledgment

This work is supported by the MEXT Grand-in-Aid for Scientific Research (B), no. 20360044, and for Exploratory Research, no. 19650003.

References

[1] L. Guibas, D. Salesin and T. Stolfi: Epsilon geometry—Building robust algorithms from imprecise computations. *Proceedings of the 5th ACM Annual Symposium on Computational Geometry.* May 1989, Saarbrücken, pp. 208–217.

[2] K. Mehlhorn and S. Naher: *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge, 1999.

[3] V. Milenkovic: Verifiable implementation of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, vol. 37 (1988), pp. 377–401.

[4] K. Sugihara: *Machine Interpretation of Line Drawings.* The MIT Press, Reading, Massachusetts, 1986.

[5] K. Sugihara: Resolvable representation of polyhedra. *Discrete and Computational Geometry*, vol. 21 (1999), pp. 243–255.

[6] K. Sugihara: Sliver-free perturbation for the Delaunay tetrahedrization. *Computer-Aided Design*, vol. 39 (2007), pp. 87–94.

[7] K. Sugihara and M. Iri: A solid modelling system free from topological inconsistency. *Journal of Information Processing*, vol. 12, no. 4 (1989), pp. 380–393.

[8] K. Sugihara and M. Iri: Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *Proceedings of the IEEE*, vol. 80, no. 9 (1992), pp. 1471–1484.

[9] C. K. Yap: Toward exact geometric computation. *Computational Geometry: Theory and Applications*, vol. 7 (1997), pp. 3–23.