35 ROBUST GEOMETRIC COMPUTATION

Chee K. Yap

INTRODUCTION

Nonrobustness refers to qualitative or catastrophic failures in geometric algorithms arising from numerical errors. Section 35.1 gives the background to these problems. Although nonrobustness is already an issue in "purely numerical" computation, the problem is compounded in "geometric computation." In Section 35.2 we characterize such computations. Researchers trying to create robust geometric software have tried two approaches: making fixed precision computation robust (Section 35.3) and making the exact approach viable (Section 35.4). There is another source of nonrobustness, and that is the phenomenon of "degenerate inputs." General methods for treating degenerate inputs are described in Section 35.5.

35.1 NUMERICAL NONROBUSTNESS ISSUES

Numerical nonrobustness in scientific computing is a well-known and wide-spread phenomenon. The root cause is the use of ***fixed precision numbers*** to represent all real numbers, usually fixed by the machine word size (e.g., 32 bits). The unpredictability of floating-point code across architectural platforms in the 1980's was resolved through a general adoption of the IEEE standard 754-1985. But this standard only makes the errors predictable and consistent across platforms; the errors are still present. None of the ad hoc methods for fixing these errors (such as treating numbers smaller than some ϵ as zero) can guarantee their elimination.

If nonrobustness is problematic in purely numerical computation, it is more intractable in "geometric" computation. In Section 35.2, we elucidate the concept of geometric computations. Based on this understanding, we conclude that nonrobustness problems in fixed precision computation cannot be solved by purely "arithmetic solutions" (better arithmetic packages, etc). Rather, a suitable ***fixed precision geometry*** is needed to substitute for the original geometry (which is usually Euclidean). We describe such approaches in Section 35.3.

In Section 35.4, we describe the ***exact approach*** for achieving robust geometric computation. This demands some type of ***big number*** package as well as further considerations.

In the final Section 35.5, we address a different but common cause of numerical nonrobustness, namely, ***data degeneracy***. Although this problem has some connection to fixed precision arithmetic, it is an issue even with the exact approach.

GLOSSARY

Fixed precision computation: A mode of computation in which every number is represented using some fixed number L of bits, usually 32 or 64. For floating point numbers, L is partitioned into $L = L_M + L_E$ for the mantissa and the exponent respectively. ***Double precision mode*** is a relaxation of fixed precision: the intermediate values are represented in $2L$ bits, but these are finally truncated back to L bits.

Nonrobustness: The property of code failing on certain kinds of inputs. Here we are mainly interested in nonrobustness that has a numerical origin: the code fails on inputs containing certain patterns of numerical values. Degenerate inputs are just extreme cases of these "bad patterns."

Benign vs. Catastrophic errors: Fixed precision numerical errors are fully expected and so are normally considered to be "benign." In purely numerical computations, errors become "catastrophic" when there is a severe loss of precision. In geometric computations, errors are "catastrophic" when the computed results are qualitatively different from the true answer (e.g., the combinatorial structure is wrong) or when they lead to unexpected or ***inconsistent*** states of the programs.

Big number packages: Software packages for representing arbitrary precision numbers (usually integers or rational numbers), and in which some basic operations on these numbers are performed exactly. For instance, $+, -, \times$ are implemented exactly with "BigIntegers." With "BigRationals," division can also be exact. Other operations such as $\sqrt{\ }$ still need approximations or rounding.

35.2 THE NATURE OF GEOMETRIC COMPUTATION

If the root cause of numerical nonrobustness is arithmetic, then it may appear that the problem can be solved with the right kind of arithmetic package. We may roughly divide the approaches into two camps, depending on whether one uses finite precision arithmetic or insists on exactness (or at least the possibility of computing to arbitrary precision). While arithmetic is an important topic on its own right, our focus here will be on "geometric" rather than "purely" arithmetic approaches for achieving robustness.

To understand why nonrobustness is especially problematic for geometric computation, we need to understand what makes a computation "geometric." Geometric computation clearly involves numerical computation, but there is something more. We use the aphorism GEOMETRIC = NUMERIC + COMBINATORIAL. But the mere presence of combinatorial structures in a numerical computation does not make the computation geometric. There must be some non-trivial ***consistency condition*** holding between the numerical data and the combinatorial data. Thus, we would not consider the classical shortest path problems on graphs to be geometric: the numerical weights assigned to edges of the graphs are not restricted by any consistency condition. Note that common restrictions on the weights (positivity, integrality, etc) are not consistency restrictions. But the related ***Euclidean shortest path problem*** (Chapter 24) is geometric. See Table 35.2.1 for further examples from well-known problems.

TABLE 35.2.1 Examples of Geometric and Non-geometric Problems

PROBLEM	GEOMETRIC?
Matrix Multiplication, Determinant	No
Hyperplane Arrangements	Yes
Shortest Paths on Graphs	No
Euclidean Shortest Paths	Yes
Point Location	Yes
Convex Hulls, Linear Programming	Yes
Minimum Circumscribing Circles	Yes

Alternatively, we can characterize a computation as "geometric" if it involves constructing or searching of a ***geometric structure*** (which may only be implicit). Informally, the incidence graph of an arrangement of hyperplanes (Chapter 21), with suitable additional labels and constraints, is an example. A geometric structure can be viewed as comprising four components:

$$D = (G, \lambda, \Phi(\mathbf{z}), I). \tag{35.2.1}$$

where $G = (V, E)$ is a directed graph, λ is a labeling function on the vertices and edges of G, Φ is the consistency predicate, and I the input parameters. Intuitively, G is the combinatorial part, λ the geometric part, and Φ constrains λ based on the structure of G. We view the input as a sequence of variables $\mathbf{z} = (z_1, \ldots, z_n)$ together with the assignment $I : \mathbf{z} \to \mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{R}^n$. The z_i's are called ***structural variables*** and the c_i's are ***parameters***. If $u \in V \cup E$, then $\lambda(u)$ is geometric in the sense that it represents a semi-algebraic set (Chapter 29) parameterized by the structural variables. More precisely, $\lambda(u)$ is a Tarski formula $\xi(\mathbf{x}, \mathbf{z})$ where $\mathbf{x} = (x_1, \ldots, x_n)$, and represents the set $\{\mathbf{a} \in \mathbb{R}^n : \xi(\mathbf{a}, \mathbf{c}) \text{ holds}\}$. We are following Tarski here in identifying semi-algebraic sets with geometric objects. The consistency relation $\Phi(\mathbf{z})$ is a Tarski formula. In practice $\Phi(\mathbf{z})$ has the form $(\forall x_1, \ldots, x_n)\phi(\lambda(u_1), \ldots, \lambda(u_m))$ where $u_1, \ldots, u_m$ ranges over elements of $V \cup E$ and ϕ can be systematically constructed from the graph G.

As an example of this notation, consider an arrangement S of hyperplanes in $\mathbb{R}^d$. The combinatorial structure $D(S)$ is the incidence graph $G = (V, E)$ between faces V of the arrangement. The parameters $\mathbf{c}$ are the coefficients of the input hyperplanes. Let the structural parameters $\mathbf{z}$ and $I : \mathbf{z} \mapsto \mathbf{c}$ be the corresponding assignment. The geometric data associates to each node v of the graph the Tarski formula $\lambda(v)$ involving $\mathbf{x}, \mathbf{z}$. When $\mathbf{c}$ are substituted for $\mathbf{z}$, then formula $\lambda(v)$ defines a face $f_{\lambda,\mathbf{c}}(v)$ (or $f(v)$ for short) of the arrangement. We use the convention that an edge $(u, v) \in E$ represents an "incidence" from $f(u)$ to $f(v)$ where dimension of $f(u)$ is one more than dimension of $f(v)$. So $f(v)$ is contained in the closure of $f(u)$. Let $\mathrm{aff}(X)$ denote the affine span of X. Then $(u, v) \in E$ implies $\mathrm{aff}(f(v)) \subseteq \mathrm{aff}(f(u))$ and $f(u)$ lies on one of the two open half-spaces in $\mathrm{aff}(f(u))$ defined by $\mathrm{aff}(f(v))$. We let $\lambda(u, v)$ be the Tarski formula involving $\mathbf{x}, \mathbf{z}$ that defines this open half-space in $\mathrm{aff}(f(u))$ that contains $f(u)$. As usual, let $f(u, v) = f_{\lambda,\mathbf{c}}(u, v)$ denote this open half- space. The consistency requirement is that (a) the set $\{f(v) : v \in V\}$ is a partition of $\mathbb{R}^d$, and (b) for each $u \in V$, the set $f(u)$ is non-empty with an irredundant representation of the form

$$f(u) = \operatorname{aff} f(u) \cap \bigcap \{f(u,v) : (u,v) \in E\}.$$

Although the above definition is complicated, all of its elements are necessary in order to capture the following additional concepts. We can suppress the parameters $\mathbf{c}$ in I and replace I by the list of structural variables $\mathbf{z}$: we call the modified quadruple

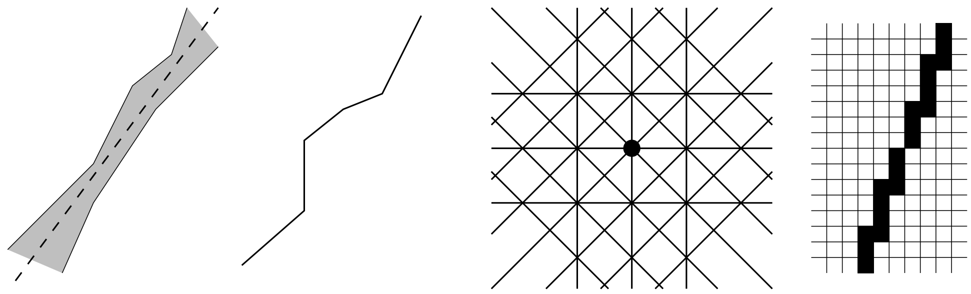
$$\widehat{D} = (G, \lambda, \Phi(\mathbf{z}), \mathbf{z})$$

an ***abstract geometric structure*** and call $D = (G, \lambda, \Phi(\mathbf{z}), I)$ an ***instance*** of $\widehat{D}$. Two geometric structures are ***structurally similar*** if they are instances of a common abstract geometric structure. The structure D in (35.2.1) is ***consistent*** if the predicate $\Phi(\mathbf{c})$ holds. Finally, an abstract geometric structure $\widehat{D}$ is ***realizable*** if there is some consistent instance of $\widehat{D}$.

35.3 FIXED PRECISION APPROACHES

This section surveys the various approaches within the fixed precision paradigm. We may classify the approaches into several basic groups. We first illustrate our classification by considering the simple question: "What is the concept of a line in fixed precision geometry?" Four basic answers to this question are illustrated in Figure 35.3.1 and in Table 35.3.1.

FIGURE 35.3.1
Finite-precision lines.



WHAT IS A FINITE PRECISION LINE?

We call the first approach ***interval geometry*** because it is the geometric analogue

TABLE 35.3.1 Concepts of a finite-precision line

	APPROACH	SUBSTITUTE FOR IDEAL LINE	SOURCE
(a)	Interval Geometry	A line fattened into a tubular region	[SS85]
(b)	Topological Distortion	A polyline	[GY86]
(c)	Rounded Geometry	A line whose equation has bounded coefficients	[Sug89]
(d)	Discretization	A suitable set of pixels	[Computer Graphics]

of interval arithmetic. Segal and Sequin [SS85] (among others) define the zone as comprising the set of points within some ε distance from the actual line.

The second approach is called ***topologically consistent distortion***. Greene and Yao [GY86] distorted their lines into polylines, where the vertices of these polylines are constrained to be at grid points. Interestingly, although the "fixed precision representation" is preserved, the number of bits used to represent these polylines can have arbitrary complexity.

The third approach follows a tact of Sugihara [Sug89]. An ideal line is specified by a linear equation, $ax + by + c = 0$. Sugihara interprets a "fixed precision line" to mean that the coefficients in this equation are integer and bounded: $|a|, |b| < K, |c| < K^2$ for some constant K. Call such lines "representable" (see Figure 35.3.1(c) for the case $K = 2$). There are $O(K^4)$ representable lines. An arbitrary line must be "rounded" to the closest (or some nearby) representable line in our algorithms. Hence we call this ***rounded geometry***.

The last approach is based on ***discretization***: in traditional computer graphics and in the pattern recognition community, a "line" is just a suitable collection of pixels. This is natural in areas where pixel images are the central objects of study, but less applicable in computational geometry, where relatively compact line representations are desired. This approach will not be considered further in this chapter.

INTERVAL GEOMETRY

In interval geometry, we "thicken" a geometric object into a "zone" containing the object. Thus a point may become a disc, and a line becomes a strip between two parallel lines: this is the simplest case and is treated by Segal and Sequin [SS85, Seg90]. They called these "toleranced objects" and in order to obtain correct predicates, they enforce ***minimum feature separations***. To do this, features that are too close must be "merged" (or "pushed apart").

Guibas, Salesin and Stolfi [GSS89] treat essentially the same class of thick objects as Segal and Sequin, although their analysis is mostly confined to geometric data based on points. Instead of insisting on minimum feature separations, their predicates are allowed to return the "don't know" truth value. Geometric predicates (called ε-predicates) for objects are systematically treated in this paper.

In general we can consider zones with non-constant descriptive complexity, e.g., a planar zone with polygonal boundaries. As with interval arithmetic, a zone is generally a conservative estimate because the precise region of uncertainty may be too complicated to compute or to maintain. In applications when zones are expand rapidly, there is danger of the zone becoming catastrophically big: Segal [Seg90] reports that a sequence of "duplicate, rotate, union" operations repeated 11 times to a cube eventually collapsed it to a single vertex.

TOPOLOGICALLY-CONSISTENT DISTORTION

The polylines that Green and Yao use as substitute for ideal lines (or segments) are determined by the presence of other lines: they construct a line arrangement that preserves certain properties (e.g., two lines intersect at a *connected* subset). They introduce the concept of a "hook": every intersection point p of two lines introduces a hook (p, q) where q is a closest grid point to p. The idea is to pull p to q. But this may generate other kinds of intersections (derived hooks) and the cascaded effects need to be carefully controlled, which they achieved in the paper. In particular, their polylines lies within a natural "zone" of the ideal line and, in some sense, every point is perturbed a distance of $\leq \sqrt{2}$. Further developments include the numerically stable algorithms in [FM91]. The interesting twist here is the use of pseudolines rather than polylines.

Hoffmann, Hopcroft and Karasick [HHK88] address the problem of intersecting polygons. Phrased in our notion of "geometric structure" (Section 35.2) their goal is to compute a combinatorial structure G that is ***consistent*** in the sense G is the structure underlying a consistent geometric structure $D = (G, \lambda, \Phi, \mathbf{c}')$. Here, $\mathbf{c}'$ need not equal the actual input parameters $\mathbf{c}$. They showed that the intersection of two polygons R_1, R_2 can be efficiently computed, i.e., a consistent G representing $R_1 \cap R_2$ can be computed. However, in their framework, $R_1 \cap (R_2 \cap R_3) \neq (R_1 \cap R_2) \cap R_3$. Hence they need to consider the triple intersection $R_1 \cap R_2 \cap R_3$. Unfortunately, this operation seems to require a non-trivial amount of geometric theorem proving ability.

This suggests that the problem of verifying consistency of combinatorial structures (the "reasoning paradigm" [HHK88]) is generally hard. Indeed, the NP-hard existential theory of reals can be reduced to such problems. But it should be noted that one can avoid this complexity trap by using weaker notions of consistency, e.g., by replacing (ideal) lines with pseudolines, as in [FM91].

STABILITY

This is a metric form of topological distortion where we place *a priori* bounds on the amount of distortion. It is analogous to backwards error analysis in numerical analysis. Framed as the problem of computing the graph G underlying some geometric structure D (as above, for [HHK88]), we could say an algorithm is ***ε-stable*** there is a consistent geometric structure $D = (G, \lambda, \Phi, \mathbf{c}')$ such that $\|\mathbf{c} - \mathbf{c}'\| < \varepsilon$ where $\mathbf{c}$ is the input parameters. We say an algorithm has ***strong*** (resp. ***linear***) stability if ε is a constant (resp., $O(n)$) where n is the input size. Fortune and Milenkovic [FM91] provides both linearly stable and strongly stable algorithms for line arrangements. Stable algorithms have been achieved for two other problems on planar point sets: maintaining a triangulation of a point set [For89], Delaunay triangulations [For92, For95a]. The latter problem can stably solved using either an incremental or a diagonal-flipping algorithm that are both $O(n^2)$ in the worst case. Sugihara and Iri [SI92] described a robust algorithm to compute the Voronoi diagram of a large set of points, but it is not proved to be stable. Jaromczk and Wasilkowski [JW94] presented stable algorithms for convex hulls.

GEOMETRIC ROUNDING

Sugihara [Sug89] shows that the problem of rounding to representable (but ideal) geometric objects can generally be reduced to the classical problem of ***simultaneous approximation by rationals***: given real numbers $a_1, \ldots, a_n$, and integer q, find integers $p_1, \ldots, p_n$ such that $\max\{|a_i q - p_i| : i = 1, \ldots, n\}$ is minimized. There are no efficient algorithms to solve this exactly, although lattice reduction techniques yield good approximations. The approach of Greene and Yao can be viewed as a geometric rounding problem, where we allow each point of the ideal line to be individually rounded to the closest grid point (but without introducing self-intersections). Milenkovic and Nackman [MN90] show that rounding a collection of disjoint simple polygons while preserving their combinatorial structure is NP-complete.

ARITHMETICAL APPROACHES

Certain approaches might be described as mainly based on arithmetic considerations (as opposed to geometric considerations). Ottmann, Thiemt and Ullrich [OTU87] show that the use of an accurate scalar product operator leads to improved robustness in segment intersection algorithms. That is, they delay the onset of qualitative errors. A case study of Dobkin and Silver [DS88] shows that permutation of operations combined with random rounding (up or down) can give accurate predictions of the total round-off error. Coupled with a multiprecision arithmetic package which is invoked when the loss in significance is too severe, they were able to improve the robustness of their code. This is one of the few studies on the loss of significance in iterated operations.

35.4 EXACT APPROACH

As the name suggests, this approach proposes to compute every geometric object "exactly". Let us summarize what this entails:

Input is exact. This may be an issue if the input is inherently approximate. Sometimes we can simply treat the approximate inputs as exact, as in the case of an input set of points without any constraints. Otherwise, there are two options: (1) to first "clean up" the inexact inputs into some exact data before feeding to the exact algorithm, or (2) to formulate a related problem in which the inexact inputs can be treated as exact (e.g., inexact input points can be viewed as centers of small balls). The cleaning up process in (1) can be non-trivial but is outside our present scope.

Numerical quantities may be implicitly represented. This is clear if we hope to compute exactly with values such as $\sqrt{2}$. Here, "explicit" representations of numbers refer to the usual positional notations (binary or decimal notations, or their use in floating-point or rational numbers).

All branching decisions in a computation are errorless. This is an important point: all "critical" phenomena in geometric computation are determined by the particular branches taken in a ***computation tree***. Hence if we make only errorless branches, the combinatorial part of a geometric structure D (see Section 35.2) will be correctly computed. Thus we only need to evaluate ***test values*** to one bit of relative precision, i.e., enough to determine the sign correctly.

For problems (such as convex hulls) requiring only rational numbers, exact computation is possible. In other applications rational arithmetic is not enough. The most general setting in which exact computation is known to be possible is the framework of ***algebraic problems*** [Yap93].

GLOSSARY

Computation Tree: A geometric algorithm in the algebraic framework can be viewed as an infinite sequence $T_1, T_2, T_3, \ldots$, of computation trees. Each T_n is restricted to inputs of size n, and is a finite tree with two kinds of nodes: (a) non-branching nodes, (b) branching nodes. Assume the input to T_n is a sequence of n real parameters $x_1, \ldots, x_n$. A non-branching node at depth i computes a value v_i, say $v_i \leftarrow f_i(v_1, \ldots, v_{i-1}, x_1, \ldots, x_n)$. A branching node tests a previous computed value v_i and makes a 3-way branch depending on the sign of v_i. Call any v_i that is used solely in a branching node a ***test value***. The branch corresponding to a zero test value is the ***degenerate branch***.

Constructible expressions: Expressions involving $+, -, \times, \div, \sqrt{\cdot}$.

Rational transformations: Transformations whose defining matrices have only rational numbers.

NAIVE APPROACH

For lack of a better term, we call the approach to exact computation in which every numerical quantity is computed exactly (explicitly if possible) the "naive approach". Thus an exact algorithm that solely relies on the use of a big number package is probably naive. In general, this approach faces the "bugbear of exact computation," namely, excessive numerical precision. For the class of "bounded-depth problems", this bugbear seems manageable [Yap93]. Still, using off-the-shelf big number package appears to be unnecessarily inefficient [FvW93, KLN91].

Recent work (cited in [YD95]) suggests that just improving current big number packages alone is unlikely to gain a factor of more than 10.

BIG EXPRESSION PACKAGES

Expression packages is the next step after number packages. This stems from the observation that the numerical part in geometric computations can usually be reduced to repeated evaluations of a few expressions (each time with different values of the variables). Examples of expressions include the distance $\sum_{i=1}^{n}(p_i - q_i)^2$ between two points p, q and determinants. If the values computed by these expressions are test values, then it is sufficient to compute them to one bit of relative precision. A technique of Clarkson [Cla92] can be used in case the test values are determinants. Some current implementation efforts are shown in Table 35.4.1.

TABLE 35.4.1 Expression Packages

SYSTEM	DESCRIPTION	REFERENCES
`LN`	Little Numbers	[FvW95]
`LEA`	Lazy Exact Numbers	[BJMM93]
`LEDA`	Library of Efficient	
	Data structures and Algorithms	[BKM+95] (Chapter 52)
`Real/Expr`	Precision-driven exact expressions	[YD95]

We briefly highlight these systems. One of `LN`'s goals is to remove all overhead associated with function calls or dynamic allocation of space for numbers with unknown sizes. It incorporates an effective floating-point filter based on static error analysis. The `LEA` system philosophy is to delay evaluating an expression until forced to, and to maintain intervals of uncertainty for values. Upon complete evaluation, the expression is discarded. The exact computation work in `LEDA` is part of a much more ambitious system of data structures for combinatorial and geometric computing. They use root bounds to achieve exactness and floating point filters for speed. The `Real/Expr` design attaches user-specified precisions to each expression, and drives this precision down to the leaves. Expression sharing and time stamps are supported. Both `LEDA` and `Real/Expr` implement exact comparisons for constructible expressions by maintaining root bounds.

Shewchuk [She96a] implemented a floating-point arithmetic package that uses adaptive-precision techniques. Although this falls short of an expression package, he shows how the underlying techniques can give adaptive evaluation of geometric predicates. For some common geometric predicates, his implementations have impressive timing.

APPLICATIONS

Burnikel, Mehlhorn and Schirra use exact techniques to compute Voronoi diagrams in [BMS94a]. They show by direct arguments that the precision needed for exact computation in this case is much better than promised by general root bounds. Chang and Milenkovic [CM93] reported their experience with using `LN` in a solid modeler. Benouamer, Michelucci and Péroche [BMP94] described applications to solid modeling (Chapter 47). Dubé and Yap [DY93] described the implementation of a big float package with dynamic error bounds, and showed its usefulness in exact computations (specifically, in Fortune's sweepline algorithm for Voronoi diagrams). Shewchuk [She96b] demonstrated the use of his adaptive bigfloat system for problems such as Delaunay triangulation. Algorithms and experiments for the exact computation of the smallest enclosing cylinder problem appear in Schömer, et al [SSTY96].

THEORY

We noted that the framework of algebraic problems has exact solutions. This class encompasses most problems in contemporary computational geometry. Algebraic problems can be solved exactly in single exponential space [Yap93]. This result is based on the recent progress on the decision problem for Tarski's language, and the associated cell decomposition problems (Chapter 29). The derivation of effective root bounds for various geometric computations is important for exact algebraic algorithms [SSTY96, BMS94a, CSY94]. The root bounds derived by Canny [Can88] are extended in [Yap96, Lecture XI]. A related topic is to develop techniques to control the cost of high precision computations. An example is the Euclidean shortest paths in 3-space (3ESP), known to be NP-hard (Chapter 24) Two approaches have been proposed towards "practical" solutions for 3ESP:

Approximation Algorithms. We would like to find a path with length within $(1+\varepsilon)$ times the length of the shortest. Here ε is an input parameter. The first solution was given by Papadimitriou [Pap85]. Choi, Sellen and Yap [CSY94] gave an alternative scheme and also analyzed this in the bit-complexity model.

Precision-sensitive Algorithm. Choi, Sellen and Yap [CSY95] introduced precision-sensitivity as the bit-complexity analogue of "output sensitive algorithms" in algebraic models of computation. They presented such an algorithm for 3ESP, and gave some experimental results.

Another issue in exact computation is the avoidance of transcendental functions or values. Rigid transformations are important in constructive solid modeling, but they are in general non-algebraic. We can get arbitrarily good approximations by using ***rational rigid transformations***. Solutions in 2 and 3 dimensions are given in [CDR92] and [MM93], respectively.

35.5 TREATMENT OF DEGENERACIES

Suppose the input to an algorithm is a set of planar points. Depending on the context, any of the following scenario might be considered "degenerate": two covertical points, three collinear points, four cocircular points. Intuitively, these are degenerate because arbitrarily small perturbations can result in qualitatively different geometric structures. Clearly this has an impact on robustness of algorithms. Degeneracy is basically a discontinuity [Yap90b, Sei94]. Sedgewick [Sed83] calls degeneracies the bugbear of geometric algorithms.

GLOSSARY

Inherent and Induced degeneracy: This is illustrated by the planar convex hull problem: an input set S is inherently degenerate if it contains three collinear points, two of which in extremal positions. If no two of the three collinear points are in extremal position, then S is not inherently degenerate. However, S may be an induced degeneracy for a divide-and-conquer algorithm because the three collinear points may appear in some subproblem of the divide-and-conquer, with two in extremal positions. Induced degeneracy is algorithm-dependent. In this chapter, we simply say "degeneracy" for induced degeneracy. More precisely, an input is ***degenerate*** if it leads to a path containing a vanishing test value in the computation tree [Yap90b]. A nondegenerate input is also said to be ***generic***.

Generic algorithm: One that is only guaranteed to be correct on generic inputs.

General algorithm: One that works correctly for all (legal) inputs. Note that "general" and "generic" are often synonymous in other literature (e.g., "generic inputs" often means inputs in "general position").

THE BASIC ISSUES

We identify three basic issues:
(a) One basic goal of this field is to provide a *systematic transformation* of a generic algorithm A into a general algorithm A'. Since generic algorithms are widespread in the literature, the availability of general tools for this $A \mapsto A'$ transformation is useful for implementing robust algorithms.
(b) Underlying any transformations $A \mapsto A'$ is some kind of perturbation of the inputs. This raises the issue of ***controlled perturbations***. E.g., if A is an algorithm for intersecting two convex polytopes, then we would like the perturbation to "expand" the input polytopes so that the incidence of a vertex in the relative interior of a face will be detected by A'.
(c) There is a ***postprocessing issue***: although A' is "correct" is some technical sense, it need not necessarily produce the same outputs as an ideal algorithm A^*. In illustration, suppose A computes the Voronoi diagram of a set of points in the plane. Four cocircular points is a degeneracy and is not treated by A. The transformed A' can handle four cocircular points but it may output two Voronoi vertices that have identical coordinates and are connected by a Voronoi edge of length 0. This may arise if we use infinitesimal perturbations. The postprocessing problem amounts to cleaning up the output of A' (removing the length-0 edges in this example) so that it conforms to the ideal output of A^*.

CONVERTING GENERIC TO GENERAL ALGORITHMS

There are essentially two general methods for this:

Blackbox Sign Evaluation Schemes. We postulate a ***sign blackbox*** that takes as input a function $f(\mathbf{x}) = f(x_1, \ldots, x_n)$ and parameters $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$, and outputs a nonzero sign (either $+$ or $-$). In case $f(\mathbf{a}) \neq 0$, this sign is guaranteed to be the sign of $f(\mathbf{a})$, but the interesting fact is that we get a nonzero sign even if $f(\mathbf{a}) = 0$. We can formulate a consistency property for the blackbox, both in an algebraic setting [Yap90b] or in a geometric setting [Yap90a]. The transformation $A \mapsto A'$ amounts to replacing all evaluation of test values by calls to this blackbox. In the paper [Yap90b] formulating this approach, a family of ***admissible schemes*** for blackboxes was given in case the functions $f(\mathbf{x})$ are polynomials.

Perturbation towards a Nondegenerate Instance. A fundamentally different approach is provided by Seidel [Sei94], based on the following idea: for any problem, if we know one non-degenerate input $\mathbf{a}^*$ for the problem, then every other input $\mathbf{a}$ can be made non-degenerate by perturbing it in the direction of $\mathbf{a}^*$. We can take the perturbed input to be $\mathbf{a} + \varepsilon\mathbf{a}^*$ for some infinitesimal ε. For example, for the convex hull of points in $\mathbb{R}^n$, we can choose $\mathbf{a}^*$ to be distinct points on the moment curve $(t, t^2, \ldots, t^n)$.

There are pros and cons to these two approaches: we currently only have blackbox schemes for polynomials (and rational functions) while Seidel's method would apply even in non-algebraic settings. On the other hand, the blackbox schemes are independent of particular problems while the nondegenerate instances $\mathbf{a}^*$ depend on the particular problem (and on the input size).

The first work in this area is the SoS ("simulation of simplicity") technique of Edelsbrunner and Mücke [EM90]. The method amounts to adding powers of an indeterminate ε to each input parameter. Such ε-methods were first used in linear programming in the 1950's. The SoS scheme (which was described for perturbing determinants) is an admissible scheme [Yap90b].

Intuitively, sign blackbox invocations should be almost as fast as the actual evaluations with high probability [Yap90b]. But its worst case behavior is exponential and this prompted Emiris and Canny to propose more efficient numerical approaches [EC95]. To each input parameter a_i in $\mathbf{a}$, it adds a perturbation $b_i\varepsilon$ (where $b_i \in \mathbb{Z}$ and ε is again an infinitesimal): these are called ***linear perturbations***. In case the test values are determinants, they show that a simple choice of b_i's will ensure non-degeneracy and efficient computation. For general rational function tests, they apply a lemma of Schwartz to show that a random choice of the b_i's will likely lead to non-degeneracy. Emiris, Canny and Seidel [ECS94, Sei94], give a general theorem for the validity of linear perturbations, and apply it to some common test polynomials.

APPLICATIONS AND PRACTICE

Michelluci [Mic95] describes implementations of blackbox schemes, based on the concept of "ε-arithmetic". One advantage of his approach is the possibility of controlling the perturbations. Experiences with the use of perturbation in the beneath-beyond convex hull algorithm in arbitrary dimensions is reported in [ECS94].

In solid modeling systems, it is very useful to systematically avoid degenerate cases (numerous in this setting). Fortune [For95b] uses symbolic perturbation to allow an "exact manifold representation" of non-regularized polyhedral solids (see Chapter 47). The idea is that dangling rectangular face (for instance) can be perturbed to look like a very flat rectangular solid, which has a manifold representation. Here, controlling the perturbation is clearly necessary.

Hertling and Weihrauch [HW94] define "levels of degeneracy" and use this to obtain lower bounds on the size of decision computation trees.

Burnikel, Mehlhorn and Schirra [BMS94b] report on the implementation of a line segment intersection algorithm and the convex hull maintenance (no deletion) algorithm in arbitrary dimensions. From this experience, they question the usefulness of perturbation methods, using three observations: (i) perturbations may increase the running time of an algorithm by an arbitrary amount, (ii) the post-processing problem can be significant, and (iii) it is not hard to directly handle degeneracies. This may appear to contradict other remarks in the literature, but on the balance, they can easily be reconciled or explained. This work serves as a reminder that (1) the development of general techniques for perturbation does not mean that direct treatment of degeneracies are no longer needed for particular applications, and (2) users must know the consequences of using these techniques. Cf. [Sch94].

35.6 OPEN PROBLEMS

1. The ε-approximate algorithms for shortest paths in [Pap85, Cla87] are intuitively reasonable, and yet they fit into neither the bit complexity nor the algebraic complexity model of computation. It is a challenge to give a proper formal model for such results (the other alternative is to convert them into results in the bit model, but this is probably non-trivial [CSY94]).
2. Can the NP-hardness result of Milenkovic and Nackman [MN90] be strengthened to show that rounding a single simple polygon is NP-hard?
3. Dobkin and Silver [DS88] is an empirical study of the complexity of iterated geometric operations: it would be nice to also have formal complexity results. In particular, does applying their "in" operation n times yield a figure that requires $\Omega(1)^n$ bits of precision (we know that $O(1)^n$ bits suffices).
4. It is known from an example of Paterson [Yap90a] that there are consistent blackboxes that do not arise from admissible orderings. So the question of constructing a family of "complete" blackbox schemes is open.

35.7 SOURCES AND RELATED MATERIAL

SURVEYS

Forrest [For87] is an influential overview of the field of computational geometry. He deplores the gap between theory and practice and describes the "open problem" of robust intersection of line segments (expressing a belief that robust solutions do not exist). Fortune treats implementation issues in geometric algorithms in [For93]. A survey of exact geometric computation is found in Yap and Dubé [YD95]. Robust geometric modelers are surveyed in [?].

RELATED CHAPTERS

Chapter 21: Arrangements
Chapter 29: Computational real algebraic geometry
Chapter 47: Solid modeling

REFERENCES

[BJMM93] M.O. Benouamer, P. Jaillon, D. Michelucci, and J-M. Moreau. A lazy arithmetic library. In *Proceedings of the IEEE 11th Symposium on Computer Arithmetic*, pages 242–269, Windsor, Ontario, June 30-July 2, 1993.

[BKM+95] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Exact geometric computation in LEDA. In *Proc. 11th ACM Sympos. Comput. Geom.*, pages C18–C19, 1995.

[BMP94] M.O. Benouamer, D. Michelucci, and B. Péroche. Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation. *Computer-Aided Design*, 26(6):403–416, June 1994.

[BMS94a] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In *Proc. 2nd European Sympos. Algorithms (ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 227–239. Springer-Verlag, 1994.

[BMS94b] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 16–23, 1994.

[Can88] J. F. Canny. *The complexity of robot motion planning.* ACM Doctoral Dissertion Awardi Series. The MIT Press, 1988. Ph.D thesis, M.I.T.

[CDR92] J.F. Canny, B. Donald, and E.K. Ressler. A rational rotation method for robust geometric algorithms. *Proc. 8th ACM Symp. on Computational Geometry*, pages 251–160, 1992. Berlin.

[Cla87] K.L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[Cla92] K.L. Clarkson. Safe and effective determinant evaluation. *IEEE Foundations of Computer Science*, 33:387–395, 1992.

[CM93] J.D. Chang and V. Milenkovic. An experiment using LN for exact geometric computations. *Proceed. 5th Canadian Conference on Computational Geometry*, pages 67–72, 1993. University of Waterloo.

[CSY94] J. Choi, J. Sellen, and C. Yap. Approximate Euclidean shortest path in 3-space. *10th ACM Symposium on Computational Geometry*, pages 41–48, 1994. Also: NYU Computer Science Technical Report No. 670, September 1994.

[CSY95] J. Choi, J. Sellen, and C. Yap. Precision-sensitive Euclidean shortest path in 3-Space. *11th ACM Symposium on Computational Geometry*, June, 1995.

[DS88] D. Dobkin and D. Silver. Recipes for Geometry & Numerical Analysis – Part I: An empirical study. *ACM Symp. on Computational Geometry*, 4:93–105, 1988.

[DY93] T. Dubé and C. K. Yap. A basis for implementing exact geometric algorithms (extended abstract), September, 1993. URL `http://cs.nyu.edu/cs/faculty/yap`.

[EC95] I. Z. Emiris and J. F. Canny. A general approach to removing degeneracies. *SIAM Journal Computing*, 24(3):650–664, 1995.

[ECS94] I. Z. Emiris, J. F. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Submitted, Algorithmica*, 1994.

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[FM91] S. J. Fortune and V. J. Milenkovic. Numerical stability of algorithms for line arrangements. *ACM Symp. on Computational Geometry*, 7:334–341, 1991.

[For87] A. R. Forrest. Computational geometry and software engineering: Towards a geometric computing environment. In D. F. Rogers and R. A. Earnshaw, editors, *Techniques for Computer Graphics*, pages 23–37. Springer-Verlag, 1987.

[For89] S. J. Fortune. Stable maintenance of point-set triangulations in two dimensions. *IEEE Foundations of Computer Science*, 30:494–499, 1989.

[For92] S. J. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations. In *Proc. 8th ACM Sympos. Comput. Geom.*, pages 83–92, 1992.

[For93] S. J. Fortune. *Progress in Computational Geometry*, chapter 3, pages 81–127. Information Geometers, 1993. Editor: R. Martin.

[For95a] S. J. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations. *Internat. J. Comput. Geom. Appl.*, 5(1):193–213, 1995.

[For95b] S. J. Fortune. Polyhedral modeling with multiprecision integer arithmetic. In *3rd ACM SIGGRAPH Symp. Solid Modeling and Applications*, pages 225–234, 1995. To appear, Computer Aided Design.

[FvW93] S. J. Fortune and C. J. van Wyk. Efficient exact arithmetic for computational geometry. *ACM Symp. on Computational Geometry*, 9:163–172, 1993.

[FvW95] S. J. Fortune and C. J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 1995. (to appear).

[GSS89] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. *ACM Symp. on Computational Geometry*, 5:208–217, 1989.

[GY86] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. *IEEE Foundations of Computer Science*, 27:143–152, 1986.

[HHK88] C. Hoffmann, J. Hopcroft, and M. Karasick. Towards implementing robust geometric computations. *ACM Symp. on Computational Geometry*, 4:106–117, 1988.

[HW94] P. Hertling and K. Weihrauch. Levels of degeneracy and exact lower complexity bounds for geometric algorithms. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 237–242, 1994.

[JW94] J.W. Jaromczyk and G.W. Wasilkowski. Computing convex hull in a floating point arithmetic. *Computational Geometry: Theory and Applications*, 4:283–292, 1994.

[KLN91] M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Trans. on Graphics*, 10:71–91, 1991.

[Mic95] D. Michelucci. An epsilon-arithmetic for removing degeneracies. In *Proceedings of the IEEE 12th Symposium on Computer Arithmetic*, pages 230–237, Windsor, Ontario, July 1995.

[MM93] V. Milenkovic and V. Milenkovic. Rational orthogonal approximations to orthogonal matrices. *Proc. 5th Canadian Conference on Computational Geometry*, pages 485–490, 1993. Waterloo.

[MN90] V. Milenkovic and L. Nackman. Finding compact coordinate representations for polygons and polyhedra. *ACM Symp. on Computational Geometry*, 6:244–252, 1990.

[OTU87] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proc. 3rd ACM Sympos. Comput. Geom.*, pages 119–125, 1987.

[Pap85] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[PCH+95] N. M. Patrikalakis, W. Cho, C.-Y. Hu, T. Maekawa, E. C. Sherbrooke, and J. Zhou. Towards robust geometric modellers, 1994 progress report. In *Proc. 1995 NSF Design and Manufacturing Grantees Conference*, pages 139–140, 1995.

[Sch94] P. Schorn. Degeneracy in geometric computation and the perturbation approach. *The Computer Journal*, 37(1):35–42, 1994.

[Sed83] R. Sedgewick. *Algorithms.* Addison-Wesley, Reading, MA, 1983.

[Seg90] M. G. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Comput. Graph.*, 24(4):105–114, August 1990.

[Sei94] R. Seidel. The nature and meaning of perturbations in geometric computing. *Comp.Geom.Theory and Applications (submitted)*, 1994.

[She96a] J. R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the Twelfth Annual Symposium on Computational, Geometry*, pages 141–150. Association for Computing Machinery, May 1996.

[She96b] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133. Association for Computing Machinery, May 1996.

[SI92] K. Sugihara and M. Iri. Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic. *Proc. IEEE*, 80(9):1471–1484, September 1992.

[SS85] M. G. Segal and C. H. Sequin. Consistent calculations for solids modelling. In *Proc. 1st ACM Sympos. Comput. Geom.*, pages 29–38, 1985.

[SSTY96] E. Schömer, J. Sellen, M. Teichmann, and C. Yap. Smallest enclosing cylinders. *ACM Symp. on Computational Geometry*, 12:C13–14, 1996. URL `http://cs.nyu.edu/cs/faculty/yap`.

[Sug89] K. Sugihara. On finite-precision representations of geometric objects. *Journal of Computer and System Sciences*, 39:236–246, 1989.

[Yap90a] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *Journal of Computer and System Sciences*, 40(1):2–18, 1990.

[Yap90b] C. K. Yap. Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computation*, 10:349–370, 1990. Proceedings, *International IFIPS Conference on System Modelling and Optimization,* Tokyo, 1987, Lecture Notes in Control and Information Science, Vol.113, pp.348-358.

[Yap93] C. K. Yap. Towards exact geometric computation. In *Fifth Canadian Conference on Computational Geometry*, pages 405–419, Waterloo, Canada, August 5–9 1993. Invited Lecture. To appear CGTA.

[Yap96] C. K. Yap. *Fundamental Problems in Algorithmic Algebra.* Princeton University Press (in press), 1996. Available on request from author and from URL `ftp://cs.nyu.edu/pub/local/yap/algebra-bk`.

[YD95] C. K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, pages 452–486. World Scientific Press, 1995. 2nd edition.