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 provides background on 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). Another source of nonrobustness is the phenomenon of degenerate inputs. General methods for treating degenerate inputs are described in Section 35.5.

41.1 NUMERICAL NONROBUSTNESS ISSUES

Numerical nonrobustness in scientific computing is a well-known and widespread phenomenon. The root cause is the use of ***fixed-precision number*** to represent real numbers, with precision 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 program behavior predictable and consistent across platforms; the errors are still present. Ad hoc methods for fixing these errors (such as treating numbers smaller than some ϵ as zero) cannot guarantee their elimination.

If nonrobustness is problematic in purely numerical computation, it apparently becomes intractable in "geometric" computation. In Section 35.2, we elucidate the concept of geometric computations. Based on this understanding, we conclude that nonrobustness problems within 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. Indeed, current research is converging on an exciting new form of computational model that we may call *guaranteed precision computation*.

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

0-8493-8524-5/97/$0.00+$.50

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.

41.2 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 in 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." Indeed, we are revisiting the age-old question *"What is Geometry?"* that has been asked and answered many times in mathematical history, by Euclid, Descartes, Hilbert, Dieudonné and others. But as in many other topics, the perspective stemming from modern computational viewpoint sheds new light. Geometric computation clearly involves numerical computation, but there is something more. We use the aphorism GEOMETRIC = NUMERIC + COMBINATORIAL to capture this. Instead of "combinatorial" we could have substituted "discrete" or sometimes "topological." What is important is that this combinatorial part is concerned with discrete relations among geometric objects. Examples of discrete relations are "a point lies on a line," "a point lies inside a simplex?," "two disks intersect." The geometric objects here are points, lines, simplices and disks. Following Descartes, each object is defined by numerical parameters. Each discrete relation is reduced to the truth of suitable numerical inequalities involving these parameters. Geometry arises when such discrete relations are used to characterize configurations of geometric objects.

The mere presence of combinatorial structures in a numerical computation does not make a computation "geometric." There must be some nontrivial ***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 41.2.1 for further examples from well-known problems.

TABLE 41.2.1 Examples of geometric and nongeometric 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 a geometric structure (which may only be implicit). The incidence graph of an arrangement of hyperplanes (Chapter 21), with suitable additional labels and constraints, is a primary example of such a structure. A ***geometric structure*** is comprised of four components:

$$D = (G, \lambda, \Phi(\mathbf{z}), I), \tag{41.2.1}$$

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 assignment. Intuitively, G is the combinatorial part, λ the geometric part, and Φ constrains λ based on the structure of G. The ***input assignment*** is $I : \{z_1, \ldots, z_n\} \to \mathbb{R}$ where the z_i's are called ***structural variables***. If $I(z_i) = c_i$ then we informally identify I with the sequence "$\mathbf{c} = (c_1, \ldots, c_n)$." The c_i's are called ***(structural) parameters***. If $u \in V \cup E$, then $\lambda(u)$ is a Tarsky formula of the form $\xi(\mathbf{x}, \mathbf{z})$ where $\mathbf{z} = (z_1, \ldots, z_n)$ are the structural variables and $\mathbf{x} = (x_1, \ldots, x_d)$. This formula defines a (Chapter 29) parameterized by the structural variables. For given $\mathbf{c}$, the semialgebraic set is $f_{\mathbf{c}}(v) = \{\mathbf{a} \in \mathbb{R}^d \mid \xi(\mathbf{a}, \mathbf{c}) \text{ holds}\}$. Folowing Tarski, we have identified semialgebraic sets in $\mathbb{R}^d$ with d-dimensional geometric objects. The consistency relation $\Phi(\mathbf{z})$ is another Tarski formula. In practice $\Phi(\mathbf{z})$ has the form $(\forall x_1, \ldots, x_d)\phi(\lambda(u_1), \ldots, \lambda(u_m))$ 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 $G = (V, E)$ of the arrangement and V is the set of faces of the arrangement. The parameter $\mathbf{c}$ consists of the coefficients of the input hyperplanes. If $\mathbf{z}$ is the corresponding structural parameters then the input assignment is $I(\mathbf{z}) = \mathbf{c}$. The geometric data

associates to each node v of the graph the Tarski formula $\lambda(v)$ involving $\mathbf{x}, \mathbf{z}$. When $\mathbf{c}$ is substituted for $\mathbf{z}$, then the formula $\lambda(v)$ defines a face $f_{\mathbf{c}}(v)$ (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 the dimension of $f(u)$ is one more than that of $f(v)$. So $f(v)$ is contained in the closure of $f(u)$. Let $\text{aff}(X)$ denote the affine span of a set $X \subseteq \mathbb{R}^d$. Then $(u, v) \in E$ implies $\text{aff}(f(v)) \subseteq \text{aff}(f(u))$ and $f(u)$ lies on one of the two open halfspaces defined by $\text{aff}(f(u))$. We let $\lambda(u, v)$ be the Tarski formula $\xi(\mathbf{x}, \mathbf{z})$ that defines the open halfspace in $\text{aff}(f(u))$ that contains $f(u)$. As usual, let $f(u, v) = f_{\mathbf{c}}(u, v)$ denote this open halfspace. 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 nonempty with an irredundant representation of the form

$$f(u) = \bigcap \{f(u, v) \mid (u, v) \in E\}.$$

Although the above definition is complicated, all of its elements are necessary in order to capture the following additional concepts. We can suppress the input assignment I, so there are only structural variables $\mathbf{z}$ (which is implicit in λ and Φ) but no parameters $\mathbf{c}$. The triple

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

becomes an ***abstract geometric structure***, and $D = (G, \lambda, \Phi(\mathbf{z}), I)$ is an ***instance*** of $\widehat{D}$. The structure D in Equation 41.2.1 is ***consistent*** if the predicate $\Phi(\mathbf{c})$ holds. An abstract geometric structure $\widehat{D}$ is ***realizable*** if it has some consistent instance. Two geometric structures D, D' are ***structurally similar*** if they are instances of a common abstract geometric structure. We can also introduce metrics on structurally similar geometric structures: if $\mathbf{c}$ and $\mathbf{c}'$ are the parameters of D, D' then define $d(D, D')$ to be Euclidean norm of $\mathbf{c} - \mathbf{c}'$.

41.3 FIXED-PRECISION APPROACHES

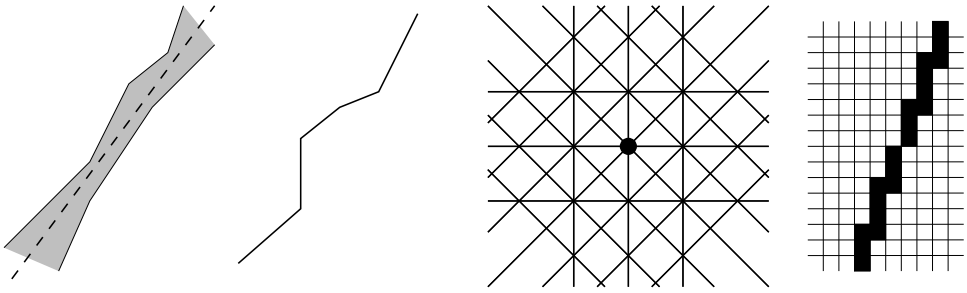
This section surveys the various approaches within the fixed-precision paradigm. Such approaches have strong motivation in the modern computing environment where fast floating point hardware has become a de facto standard in every computer. If we can make our geometric algorithms robust within machine arithmetic, we are assured of the fastest possible implementation. 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 41.3.1 and in Table 41.3.1.

WHAT IS A FINITE-PRECISION LINE?

We call the first approach ***interval geometry*** because it is the geometric analogue of interval arithmetic. Segal and Sequin [SS85] and others define a zone surrounding the line composed of all 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

FIGURE 41.3.1
Four concepts of finite-precision lines.



polylines are constrained to be at grid points. Note that although the "fixed-precision representation" is preserved, the number of bits used to represent these polylines can have arbitrary complexity.

TABLE 41.3.1 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

The third approach follows a tack 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 41.3.1(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 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 disk, 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 nonconstant 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 where zones expand rapidly, there is danger of the zone becoming catastrophically large: Segal [Seg90] reports that a sequence of duplicate-rotate-union operations repeated eleven times to a cube eventually collapsed it to a single vertex.

TOPOLOGICALLY-CONSISTENT DISTORTION

Sugihara and Iri [SI89b, SIII00] advocates an approach based on preserving topological consistency. These ideas have been applied to several problems, including geometric modeling [SI89a] and Voronoi diagrams for point sets [SI92]. In their approach, one first chooses some topological property (e.g., planarity of the underlying graph) and construct geometric algorithms that preserve the chosen property. Two difficulties in this prescription are (1) how to choose appropriate topological properties, and (2) in what sense does this "work"? Greene and Yao consider the problem of maintaining certain "topological properties" of an arrangement of finite-precision line segments. They introduce polylines as substitutes for ideal line segments in order to preserve certain properties of ideal arrangements (e.g., two line segments intersect in a *connected* subset). Each polyline is a distortion of an ideal segment σ when constrained to pass through the "hooks" of σ (i.e., grid points nearest to the intersections of σ with other line segments). But this may generate new intersections (derived hooks) and the cascaded effects must be carefully controlled. The grid model of Greene-Yao has been taken up by several other authors [Hob99, GM95, GGHT97]. Extension to higher dimensions is harder: there is a solution of Fortune [For98] in 3-dimension. 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 in a consistent way. Phrased in terms of our notion of "geometric structure" (Section 35.2) their goal is to compute a combinatorial structure G that is *consistent* in the sense that G is the structure underlying a consistent geometric structure $D = (G, \lambda, \Phi, \mathbf{c}')$. Here, $\mathbf{c}'$ need not equal the actual input parameter vector $\mathbf{c}$. They show 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 nontrivial amount of geometric theorem proving ability.

This suggests that the problem of verifying consistency of combinatorial struc-

tures (the "reasoning paradigm" [HHK88]) is generally hard. Indeed, the NP-hard existential theory of reals can be reduced to such problems. In some sense, the ultimate approach to ensuring consistency is to design "parsimonious algorithms" in the sense of Fortune [For89]. This also amounts to theorem proving as it entails deducing the consequences of all previous decisions along a computation path.

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*** if there is a consistent geometric structure $D = (G, \lambda, \Phi, \mathbf{c}')$ such that $\|\mathbf{c} - \mathbf{c}'\| < \epsilon$ where $\mathbf{c}$ is the input parameter vector. 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] provide 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], and Delaunay triangulations [For92, For95a]. The latter problem can be solved stably using either an incremental or a diagonal-flipping algorithm that is $O(n^2)$ in the worst case. Jaromczk and Wasilkowski [JW94] presented stable algorithms for convex hulls. Stability is a stronger requirement than topological consistency. E.g., the topological algorithms (e.g., [SI92]) have not been proven stable.

ROUNDED GEOMETRY

Sugihara [Sug89] shows that the above problem of "rounding a line" can be reduced to the classical problem of *simultaneous approximation by rationals*: given real numbers $a_1, \ldots, a_n$, find integers $p_1, \ldots, p_n$ and q such that $\max_{1 \le i \le n} |a_i q - p_i|$ is minimized. There are no efficient algorithms to solve this exactly, although lattice reduction techniques yield good approximations. The above approach of Greene and Yao can also be viewed as a geometric rounding problem. The "rounded lines" in the Greene-Yao sense is a polyline with unbounded combinatorial complexity; but rounded lines in the Sugihara sense still have constant complexity. Milenkovic and Nackman [MN90] show that rounding a collection of disjoint simple polygons while preserving their combinatorial structure is NP-complete. In Section 35.5, rounded geometry is seen in a different light.

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, the onset of qualitative errors is delayed. 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. By coupling this with a multiprecision arithmetic package that is invoked when the loss in significance is too severe, they are able to

improve the robustness of their code. There is a large literature on computation under the interval arithmetic model (e.g., [Ull90]). It is related to what we call interval geometry above. There are also systems providing programming language support for interval analysis.

41.4 EXACT APPROACH

As the name suggests, this approach proposes to compute without any error. The initial interpretation is that every numerical quantity is computed exactly. While this has an natural meaning when all numerical quantities are rational, it is not obvious what this means for values such as $\sqrt{2}$ which cannot be exactly represented "explicitly." Informally, a number representation is explicit if it facilitates efficient comparison operations. In practice, this amounts to representing numbers by one or more integers in some positional notation (this covers the usual representation of rational numbers as well as floating point numbers). Although we could achieve numerical exactness in some modified sense, this turns out to be unnecessary. The solution to the nonrobustness only requires a weaker notion of exactness: it is enough to ensure "geometric exactness." In the "Geometric = Numeric + Combinatorial" formulation, the exactness is not to be found in the numeric part, but in the combinatorial part, as this encodes the geometric relations. Hence this approach is called ***Exact Geometric Computation*** (EGC), and it entails the following:

Input is exact. We cannot speak of exact geometry unless this is true. This assumption can be an issue if the input is inherently approximate. Sometimes we can simply treat the approximate inputs as "nominally" exact, as in the case of an input set of points without any constraints. Otherwise, there are two options: (1) "clean up" the inexact input, by transforming it to data that is exact; or (2) formulate a related problem in which the inexact input can be treated as exact (e.g., inexact input points can be viewed as the *exact* centers of small balls). So the convex hull of a set of points becomes the convex hull of a set of balls. The cleaning up process in (1) may be nontrivial as it may require perturbing the data to achieve some consistency property and lies outside our present scope. The transformation (2) typically introduces a computationally harder problem. Not much research is currently available for such transformed problems. In any case, (1) and (2) still end up with exact inputs for a well-defined computational problem.

Numerical quantities may be implicitly represented. This is necessary if we want to represent irrational values exactly. In practice, we will still need explicit numbers for various purposes (e.g., comparison, output, display, etc). So a corollary is that numerical approximations will be important, a remark that was not obvious in the early days of EGC.

All branching decisions in a computation are errorless. At the heart of EGC is the idea that all "critical" phenomena in geometric computations are determined by the particular sequence branches taken in a *computation tree*. The key observation is that the sequence of branching decisions completely decides the combinatorial nature of the output. Hence if we make only errorless

branches, the combinatorial part of a geometric structure D (see Section 35.2) will be correctly computed. To ensure this, 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* [Yap97].

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) nonbranching nodes, (b) branching nodes. Assume the input to T_n is a sequence of n real parameters $x_1, \ldots, x_n$. A nonbranching 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. In case v_i is a complex value, we simply that the sign of the real part 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***.

Exact Geometric Computation (EGC): Preferred name for the general approach of "exact computation," as it accurately identifies the goal of determining geometric relations exactly. The exactness of the computed numbers is either unnecessary, or should be avoided if possible.

Composite Precision Bound: This is specified by a pair $[r, a]$ where $r, a \in \mathbb{R} \cup \{\infty\}$. For any $z \in \mathbb{C}$, let $z[r, a]$ denote the set of all $\widetilde{z} \in \mathbb{C}$ such that $|z - \widetilde{z}| \leq \max\{2^{-a}, |z|2^{-r}\}$. When $r = \infty$, then $z[\infty, a]$ comprises all the numbers $\widetilde{z}$ that approximates z with an absolute error of 2^{-a}; we say this approximation $\widetilde{z}$ has a ***absolute bits***. Similarly, $z[r, \infty]$ comprises all numbers $\widetilde{z}$ that approximates z with a relative error of 2^{-r}; we say this approximation $\widetilde{z}$ has r ***relative bits***.

Constant Expressions: Let Ω be a set of complex algebraic operators; each operator $\omega \in \Omega$ is a partial function $\omega : \mathbb{C}^{a(\omega)} \to \mathbb{C}$ where $a(\omega) \in \mathbb{N}$ is the arity of ω. If $a(\omega) = 0$, then ω is identified with a complex number. Let $\mathcal{E}(\Omega)$ be the set of expressions over Ω where an expression E is a rooted DAG (directed acyclic graph) and each node with outdegree $n \in \mathbb{N}$ is labeled with an operator of Ω of arity n. There is a natural ***evaluation function*** $\mathrm{val} : \mathcal{E}(\Omega) \to \mathbb{R}$. If Ω has partial functions, then val() is also partial. If $\mathrm{val}(E)$ is undefined, we write $\mathrm{val}(E) = \uparrow$ and say E is ***invalid***. When $\Omega = \Omega_2 = \{+, -, \times, \div, \sqrt{}\} \cup \mathbb{Z}$ we get the important class of ***constructible expressions***, so-called because their values are precisely the constructible reals.

Constant Zero Problem, $\mathrm{ZERO}(\Omega)$***:*** Given $E \in \mathcal{E}(\Omega)$, decide if $\mathrm{val}(E) = \uparrow$; if not, decide if $\mathrm{val}(E) = 0$.

Guaranteed Precision Evaluation Problem, $\mathrm{GVAL}(\Omega)$***:*** Given $E \in \mathcal{E}(\Omega)$ and $a, r \in \mathbb{Z} \cup \{\infty\}$, $(a, r) \neq (\infty, \infty)$, compute some approximate value in $\mathrm{val}(E)[r, a]$,

Schanuel's Conjecture: If $z_1, \ldots, z_n \in \mathbb{C}$ are linearly independent over $\mathbb{Q}$, then the set $\{z_1, \ldots, z_n, e^{z_1}, \ldots, e^{z_n}\}$ contains a subset $B = \{b_1, \ldots, b_n\}$ that is algebraically independent, i.e., there is no polynomial $P(X_1, \ldots, X_n) \in \mathbb{Q}[X_1, \ldots, X_n]$

such that $P(b_1, \ldots, b_n) = 0$. This conjecture generalizes several deep results in transcendental number theory, and implies many other conjectures.

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 relies solely on the use of a big number package is probably naive. This approach, even for rational problems, faces the "bugbear of exact computation," namely, high numerical precision. Using an off-the-shelf big number package does not appear to be a practical option [FvW93a, KLN91, Yu92]. There is evidence (surveyed in [YD95]) that just improving current big number packages alone is unlikely to gain a factor of more than 10.

BIG EXPRESSION PACKAGES

The most common examples of expressions are determinants and the distance $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$ between two points p, q. A big expression package allows a user to construct and evaluate expressions with big numbers values. They represent the next logical step after big number packages, and are motivated by the observation that the numerical part of a geometric computation is invariably reduced to repeated evaluations of a few variable[1] expressions (each time with different constants substituted for the variables). When these expressions are test values, then it is sufficient to compute them to one bit of relative precision. Some implementation efforts are shown in Table 41.4.1.

TABLE 41.4.1 Expression packages.

SYSTEM	DESCRIPTION	REFERENCES
`LN`	Little Numbers	[FvW96]
`LEA`	Lazy ExAct Numbers	[BJMM93]
`Real/Expr`	Precision-driven exact expressions	[YD95]
`LEDA Real`	Exact numbers of Library of Efficient Data structures and Algorithms	[BFMS99, BKM+95]
`Core Library`	Package with Numerical Accuracy API and C++ interface	[KLPY99]

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 experience in [CM93] suggests that `LN`'s approach is too aggressive as it leads to code bloat. The `LEA` system philosophy is to delay evaluating an expression until forced to, and to main-

[1] These expressions involves variables, unlike the constant expressions in $\mathcal{E}(\Omega)$.

tain intervals of uncertainty for values. Upon complete evaluation, the expression is discarded. It uses root bounds to achieve exactness and floating point filters for speed. The `Real/Expr Package` is the first system to achieve guaranteed precision for a general class of non-rational expressions. Its introduces the "precision-driven mechanism" whereby a user-specified precision at the root of the expression is transformed and downward-propagated towards the leaves, while approximate values generated at the leaves are evaluated and error bounds upward-propagated up to the root. This upward-downward process may need to be iterated. `LEDA Real` is a number type with a similar mechanism. It is part of a much more ambitious system of data structures for combinatorial and geometric computing (see Chapter 65). The semantics of `Real/Expr` of expression assignment is akin to constraint propagation in the constraint programming paradigm. The `Core Library` (`CORE`) is derived from `Real/Expr` with the goal of making the system as easy to use as possible. The two pillars of this transformation is the adoption of conventional assignment semantics, and the introduction of a simple ***Numerical Accuracy API*** [Yap98].

The `CGAL` Library (Chapter 65) is a major library of geometric algorithms which are designed according to the EGC principles. While it has some native number types supporting rational expressions, the current distribution relies on `LEDA Real` or `CORE` for more general algebraic expressions. Shewchuk [She96] implements an arithmetic package that uses adaptive-precision floating-point representations. While not a big expression package, it has been used to implement polynomial predicates and shown to be extremely efficient.

THEORY

The class of algebraic computational problems encompasses most problems in contemporary computational geometry. Such problems can be solved exactly in singly-exponential space [Yap97]. This general result is based on recent progress in the decision problem for Tarski's language, on the associated cell decomposition problems, as well as cell adjacency computation (Chapter 32). However, general EGC libraries such as `Core Library` and `LEDA Real` depend directly on the algorithms for the guaranteed precision evaluation problem GVAL(Ω) (see Glossary), where Ω is the set of operators in the computation model. The possibility of such algorithms can be reduced to the recursiveness of a constellation of problems that might be called the ***Fundamental Problems of EGC***. First is the Constant Zero Problem ZERO(Ω). But there are two closely related problems. In the ***Constant Validity Problem*** VALID(Ω), we are to decide if a given $E \in \mathcal{E}(\Omega)$ is valid, i.e., $\text{val}(E) \neq \uparrow$. The ***Constant Sign Problem*** SIGN(Ω) is to compute $\texttt{sign}(E)$ for any given $E \in \mathcal{E}(\Omega)$, where $\texttt{sign}(E) \in \{\uparrow, -1, 0, +1\}$. In case $\text{val}(E)$ is complex, define $\texttt{sign}(E)$ to be the sign of the real part of $\text{val}(E)$.

There is a natural hierarchy of the expression classes, each corresponding to a class of complex numbers as shown in 41.4.2. In Ω_3, $P(X)$ is any polynomial with integer coefficients and I is some means of identifying a unique root of $P(X)$: I may be an complex interval bounding a unique root of $P(X)$, or an integer i to indicate the ith largest real root of $P(X)$. The operator $\text{RootOf}(P, I)$ can be generalized to allow allowing expressions as coefficients of $P(X)$ as in Burnikel et al. [BFM+01], or by introducing systems of polynomial equations as in Richardson [Ric97]. Although Ω_4 can be treated as a set of real operators, it is more natural to

TABLE 41.4.2 Expression Hierarchy.

OPERATORS	NUMBER CLASS	EXTENSIONS
$\Omega_0 = \{+, -, \times\} \cup \mathbb{Z}$	Integers	
$\Omega_1 = \Omega_0 \cup \{\div\}$	Rational Numbers	$\Omega_1^+ = \Omega_1 \cup \mathbb{Q}$
$\Omega_2 = \Omega_1 \cup \{\sqrt{\cdot}\}$	Constructible Numbers	$\Omega_2^+ = \Omega_2 \cup \{\sqrt[k]{\cdot} : k \geq 3\}$
$\Omega_3 = \Omega_2 \cup \{\text{RootOf}(P(X), I)\}$	Algebraic Numbers	Use of $\diamond(E_1, \ldots, E_d, i)$, [BFM$^+$01]
$\Omega_4 = \Omega_3 \cup \{\exp(\cdot), \ln(\cdot)\}$	Elementary Numbers (cf. [Cho99])	

treat Ω_4 (and sometimes Ω_3) as complex operators. Thus the elementary functions $\sin x, \cos x, \arctan x$, etc, are available as expressions in Ω_4.

It is clear ZERO(Ω) and VALID(Ω) is reducible to SIGN(Ω). For Ω_4, all three problems are recursively equivalent. The fundamental problems related to Ω_i is decidable for $i \leq 3$. It is a major open question whether the fundamental problems for Ω_4 are decidable. These questions have been studied by Richardson and others [Ric97, Cho99, MW96]. The most general positive result is that SIGN(Ω_3) is decidable. An intruiguing conditional result is that ZERO(Ω_4) is decidable if Schanuel's conjecture is true; this may be deduced from Richardson's work [Ric97].

CONSTRUCTIVE ROOT BOUNDS

In practice, algorithms for the guaranteed precision problem GVAL(Ω_3) can exploit the fact that algebraic numbers have computable root bounds. An ***root bound*** for Ω is a total function $\beta : \mathcal{E}(\Omega) \to \mathbb{R}_{\geq 0}$ such that for all $E \in \mathcal{E}(\Omega)$, if E is valid and $\text{val}(E) \neq 0$ then $|\text{val}(E)| \geq \beta(E)$. More precisisely, β is called an ***exclusion*** root bound; it is an ***inclusion root bound*** when the inequality becomes "$|\text{val}(E)| \leq \beta(E)$." We use the (exclusion) root bound β to solve ZERO(Ω) as follows: to test if an expression E evaluates to zero, we compute an approximation α to $\text{val}(E)$ such that $|\alpha - \text{val}(E)| < \beta(E)/2$. While computing α, we can recursively verify the validity of E. If E is valid, we compare α with $\beta/2$. It is easy to conclude that $\text{val}(E) = 0$ if $|\alpha| \leq \beta/2$. Otherwise $|\alpha| > \beta/2$, and the sign of $\text{val}(E)$ is that of α. An important remark is that the root bound β determines the worst-case complexity. This is unavoidable if $\text{val}(E) = 0$. But if $\text{val}(E) \neq 0$, the worst case may be avoided by iteratively computing α_i with increasing absolute precision ε_i. If for any $i \geq 1$, $|\alpha_i| > \varepsilon_i$, we stop and conclude $\texttt{sign}(\text{val}(E)) = \texttt{sign}(\alpha_i) \neq 0$.

There is an extensive classical mathematical literature on root bounds, but they are usually not suitable for computation. Recently, new root bounds have been introduced that explicitly depend on the structure of expressions $E \in \mathcal{E}(E)$. In [LY01], such bounds are called ***constructive*** in the following sense: (i) There are easy-to-compute recursive rules for maintaining a set of numerical parameters $u_1(E), \ldots, u_m(E)$ based on the structure of E, and (ii) $\beta(E)$ is given by an explicit formula in terms of these parameters. The first constructive bounds in EGC were the degree-length and degree-height bounds of Yap and Dubé [YD95, Yap00] in their implementation of `Real/Expr`. The (Mahler) Measure Bound was introduced even earlier by Mignotte [Mig82, BFMS00] for the problem of "identifying algebraic numbers." A major improvement was achieved with the introduction of the BFMS

Bound [BFMS00]. Li-Yap [LY01] introduced another bound aimed at improving the BFMS Bound in the presence of division. Comparison of these bounds is not easy: but let us say a bound β ***dominates*** another bound β' if for every $E \in \mathcal{E}(\Omega_2)$, $\beta(E) \leq \beta'(E)$. Burnikel et al. [BFM$^+$01] generalized the BFMS Bound to the BFMSS Bound. Yap noted that if we incorporate a symmetrizing trick for the $\sqrt{x/y}$ transformation, then BFMSS will dominate BFMS. Among current constructive root bounds, three are not dominated by other bounds: BFMSS, Measure, and Li-Yap Bounds. In general, BFMSS seems to be the best. Other root bounds include a multivariate root bound of Canny [Can88] (see extension in [Yap00, Chapter XI]) and an Eigenvalue Bound of Scheinerman [Sch00]. A recent factoring technique of Pion and Yap [PY03] can be used to improve the existing bounds (in particular, BFMSS). This technique can exploit the presence of k-ary input numbers, and is thus favorable for the majority of realistic inputs (which are binary or decimal).

FILTERS

An extremely effective technique for speeding up predicate evaluation is based on the filter concept. Since evaluating the predicate amounts to determining the sign of an expression E, we can first use machine arithmetic to quickly compute an approximate value α of E. For a small overhead, we can simultaneously determine an error bound ε where $|\text{val}(E) - \alpha| \leq \varepsilon$. If $|\alpha| > \varepsilon$, then the sign of α is the correct one and we are done. Otherwise, we evaluate the sign of E again, this time using a sure-fire if slow evaluation method. The algorithm used in the first evaluation is called a (floating-point) ***filter***. The expected cost of the two-stage evaluation is small if the filter is efficient with a high probability of success. This idea was first used by Fortune and van Wyk [FvW96]. Floating-point filters can be classified along the static-to-dynamic dimension: ***static filters*** compute the bound ε solely from information that are known at compile time while ***dynamic filters*** depend on information available at run time. There is an ***efficiency-efficacy tradeoff***: static filters (e.g., FvW Filter [FvW96]) are more efficient, but dynamic filters (e.g., BFS Filter [BFS98]) are more accurate (efficacious). Interval arithmetic has been shown to be an effective way to implement dynamic filters [BBP01]. Automatic tools for generating filter code are treated in [FvW93b, Fun97]. Filters can been elaborated in several ways. First, we can use a cascade of filters [BFS98]. The "steps" of an algorithm which are being filtered can be defined at different levels of granularity. One extreme is to consider an entire algorithm as one step [MNS$^+$96, KW98]. A general formulation "structural filtering" is proposed in [FMN99]. Probabilistic analysis [DP99] shows the efficacy of arithmetic filters. The filtering of determinants is treated in several papers [Cla92, BBP01, PY01, BY00].

Filtering is related to program checking [BK95, BLR93]. View a computational problem P as an input-output relation, $P \subseteq I \times O$ where I, O is the input and output spaces respectively. Let be A a (standard) ***algorithm*** for P which, viewed as a total function $A : I \to O \cup \{NaN\}$, has the property that for all $i \in I$, $(i, A(i)) \in P$ iff there is some $o \in O$ such that $(i, o) \in P$. Let $H : I \to O \cup \{NaN\}$ be another algorithm with no restrictions; call H a ***heuristic algorithm*** for P. Let $F : I \times O \to \{true, false\}$. Then F is ***checker*** for P if F computes the characteristic function for P, $F(i, o) = true$ iff $(i, o) \in P$. Note that F is a checker for the problem P, and not for any purported program for P. Hence, unlike program checking, we do not require any special properties of P such as self-reducibility. We

call F a ***filter*** for P if $F(i,o) = true$ implies $(i,o) \in P$. So filters are less restricted than checkers. A ***filtered program*** for P is therefore a triple (H, F, A) where H is heuristic algorithm, A a standard algorithm and F a filter. To run this program on input i, we first compute $H(i)$ and check if $F(i, H(i))$ is true. If so, we output $H(i)$; otherwise compute and output $A(i)$. Filtered programs can be extremely effective when H, F are both efficient and efficacious. Usually H is easy—it is just a machine arithmetic implementation of an exact algorithm. The filter F can be more subtle, but it is still more readily constructed than any checker. The problem P_{sdet} of computing the sign of determinants illustrates this: the only checkers we know here is trivial, amounting to computing the determinant itself. On the other hand, effective filters for P_{sdet} are known [BBP01, PY01].

PRECISION COMPLEXITY

An important goal of EGC is to control the cost of high-precision computation. We describe two approaches based on modifying the algorithmic specification.

In predicate evaluation, there is an in-built precision of 1-relative bit (this precision guarantees the correct sign in the predicate evaluation). But in construction steps, any precision guarantees must be explicitly requested by the user. For optimization problems, a standard method to specify precision is to incorporate an extra input parameter $\epsilon > 0$. Assume the problem is to produce an output x to minimizes the function $\mu(x)$. An ***ϵ-approximation algorithm*** will output a solution x such that $\mu(x) \leq (1+\varepsilon)\mu(x^*)$ for some optimum x^*. An example is the ***Euclidean Shortest-path Problem in 3-space*** (3ESP). Since this problem is NP-hard (Section 24.5), we seek an ϵ-approximation algorithm. A simple way to implement an ϵ-approximation algorithm is to directly implement any *exact* algorithm in which the underlying arithmetic has guaranteed precision evaluation (using, e.g., `Core Library`). However, the bit complexity of such an algorithm may not be obvious. The more conventional approach is to explicitly build the necessary approximation scheme directly into the algorithm. One such scheme was given by Papadimitriou [Pap85] which is polynomial time in n and $1/\varepsilon$. Choi et al. [CSY97] give an improved scheme, and perform a rare bit-complexity analysis.

Another way to control precision is to consider output complexity. In geometric problems, the input and output ***sizes*** are measured in two independent ways: combinatorial size and bit sizes. Let the input combinatorial and input bit sizes be n and L, respectively. By an L-bit input, we mean each of the numerical parameters in the description of the geometric object (see Section 35.2) is an L-bit number. Now an extremely fruitful concept in algorithmic design is this: an algorithm is said to be ***output-sensitive*** if the complexity of the algorithm can be made a function of the output size as well as of the input size parameters. In the usual view of output-sensitivity, only the output combinatorial size is exploited. Choi et al. [SCY00] introduced the concept of ***precision-sensitivity*** to remedy this gap. They presented the first precision-sensitive algorithm for 3ESP, and gave some experimental results. Using the framework of ***pseudo-approximation algorithms***, Asano et al. [AKY02] gave new precision-sensitive algorithms for 3ESP, as well as for an optimal d_1-motion for a rod.

GEOMETRIC ROUNDING

We saw rounded geometry as one of the fixed-precision approaches (Section 35.3) to robustness. But geometric rounding is also important in EGC, with a difference. The EGC problem is to "round" a geometric structure (Section 35.2) D to a geometric structure D' with lower precision. In fixed-precision computation, one is typically asked to construct D' from some input S that *implicitly* defines D. In EGC, D is explicitly given (e.g., D may be computed from S by an EGC algorithm). The EGC view should be more tractable since we have separated the two tasks: (a) computing D and (b) rounding D. We are only concerned with (b), the ***pure rounding problem***. For instance, if S is a set of lines that are specified by linear equations with L-bit coefficients, then the arrangement $D(S)$ of S would have vertices with $2L + O(1)$-bit coordinates. We would like to round the arrangement, say, back to L bits. Such a situation, where the output bit precision is larger than the input bit precision, is typical. If we pipeline several of these computations in a sequence, the final result could have a very high bit precision unless we perform rounding.

If D rounds to D', we could call D' a ***simplification*** of D. This viewpoint makes connection to a larger literature on simplification of geometry (e.g., simplifying geometric models in computer graphics and visualization (Chapter 54). Two distinct objectives goals in simplification are ***combinatorial*** versus ***precision simplification***. For example, a problem that has been studied in a variety of contexts (e.g., Douglas-Peucker algorithm in computational cartography) is that of simplifying a polygonal line P. We can use ***decimation*** to reduce the combinatorial complexity (i.e., number of vertices $\#(P)$), for example, by omitting every other vertex in P. Or we can use ***clustering*** to reduce the bit-complexity of P to L-bits. E.g., we collapse all vertices that lie within the same grid cell, assuming grid points are L-bit numbers. Let $d(P, P')$ be the Hausdorff distance between P and another polyline P'; other similar measures of distance may be used. In any simplification P' of P, we want to keep $d(P, P')$ small. In [BCD$^+$02], two optimization problems are studied: in the ***Min-# Problem***, given P and ε, find P' to minimize $\#(P)$, subject to $d(P, P') \leq \varepsilon$. In the ***Min-$\varepsilon$ Problem***, the roles of $\#(P)$ and $d(P, P')$ are reversed. For EGC applications, optimality can often be relaxed to simple feasibility. Path simplification can be generalized to the simplification of any cell complexes.

BEYOND ALGEBRAIC

Non-algebraic computation over Ω_4 is important in practice. This includes the use of elementary functions such as $\exp x, \ln x, \sin x$, etc, which are found in standard libraries (`math.h` in `C/C++`). Elementary functions can be implemented via their representation as ***hypergeometric functions***, an approach taken Du et al. [DEMY02]. They described solutions for fundamental issues such as automatic error analysis, hypergeometric parameter processing and argument reduction. If f is a hypergeometric function and x is an explicit number, one can compute $f(x)$ to any desired absolute accuracy. But in the absence of root bounds for Ω_4, we cannot solve the guaranteed precision problem GVAL(Ω_4). One systematic way to get around this is to invoke the uniformity conjecture [Ric00]: this conjecture provides us with a bound. If this bound ever lead to an error, we would have produced a counterexample to the uniformity conjecture.

There are situations where we can either avoid the use of transcendental func-

tions, or their apparent need turn out to be non-essential (e.g., in motion planning). For instance, rigid transformations are important in solid modeling, but they involve trigonometric functions. We can get arbitrarily good approximations by using *rational rigid transformations*. Solutions in 2 and 3 dimensions are given by Canny et al. [CDR92] and Milenkovic and Milenkovic [MM93], respectively.

APPLICATIONS

We now consider issues in implementing specific algorithms under the EGC paradigm. The rapid growth in the number of such algorithms means the following list is quite partial. We attempt to illustrate the range of activities in several groups: **(i)** The early EGC algorithms produced were those that are easily reduced to integer arithmetic and polynomial predicates, such convex hulls or Delaunay triangulations. The goal was to demonstrate that such algorithms are implementable and relatively efficient (e.g., [FvW96]). To treat irrational predicates, the careful analysis of root bounds were needed to ensure efficiency. Thus, Burnikel, Mehlhorn, and Schirra [BMS94, Bur96] gave sharp bounds in the case of Voronoi diagrams for line segments. Similarly, Dubé and Yap [DY93] analyzed the root bounds in Fortune's sweepline algorithm, and first identified the usefulness of floating point approximations in EGC. Another approach is to introduce algorithms that use new predicates with low algebraic degrees. This line of work was initiated by Liotta, Preparata and Tamassia [LPT97, BS00]. **(ii)** Polyhedral modeling is a natural domain for EGC techniques. Two efforts are [CM93, For97]. The most general viewpoint here uses Nef polyhedra [See01] in which open, closed or half-open polyhedral sets are represented. This is a radical departure from the traditional solid modeling based on ***regularized sets*** and the associated ***regularized operators***. The regularization of a set $S \subseteq \mathbb{R}^d$ is obtained as the closure of the interior of S; regularized sets do not allow lower dimensional features. E.g., a line sticking out of a solid is not permitted. Treatment of Nef polyhedra was previously impossible outside the EGC framework. **(iii)** An interesting domain is optimization problems such as linear and quadratic programming [Gae99, GS00] and smallest enclosing cylinder problem [SSTY00]. In Linear Programming, there is a tradition of using benchmark problems for evaluating algorithms and their implementations. But what is lacking in the benchmarks is ***reference solutions*** with guaranteed accuracy to (say) 16 digits. One application of EGC algorithms to to produce such solutions. **(iv)** An area of major challenge is computation of algebraic curves and surfaces. Krishnan et al. [KFC$^+$01] implemented a library of algebraic primitives to support the manipulation of algebraic curves. Algorithms for low degree curves and surfaces are beginning to be addressed, e.g., [BEH$^+$02, GHS01, Wei02]. **(v)** The development of general geometric libraries such as `CGAL` [HHK$^+$01] or LEDA [MN95] exposes a range of issues peculiar to EGC. For instance, in EGC we want a framework where various number kernels and filters can be used for a single algorithm.

41.5 TREATMENT OF DEGENERACIES

Suppose the input to an algorithm is a set of planar points. Depending on the context, any of the following scenarios might be considered "degenerate": two cover-

tical points, three collinear points, four cocircular points. Intuitively, these are degenerate because arbitrarily small perturbations can result in qualitatively different geometric structures. Degeneracy is basically a discontinuity [Yap90b, Sei94]. Sedgewick [Sed83] calls degeneracies the "bugbear of geometric algorithms." Degeneracies is a major cause nonrobustness for two reasons. First, it present severe difficulties for approximate arithmetic. Second, even under the EGC paradigm, implementors is faced with a large number of special degenerate cases that must be treated (this number grows exponentially in the dimension of the underlying space). Thus the need to develop general techniques for handling degeneracies.

GLOSSARY

Inherent and induced degeneracy: This is illustrated by the planar convex hull problem: an input set S with three collinear points p, q, r is inherently degenerate if it lies entirely in one halfplane determined by the line through p, q, r. If p, q, r are collinear but S does not lie on one side of the line through p, q, r, then we may have an induced degeneracy for a divide-and-conquer algorithm. This happens when the algorithm solves a subproblem $S' \subseteq S$ containing p, q, r with all the remaining points on one side. 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 used synonymously in other literature (e.g., "generic inputs" often means inputs in general position).

THE BASIC ISSUES

1. 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.

2. Underlying any transformations $A \mapsto A'$ is some kind of perturbation of the inputs. This raises the issue of *controlled perturbations*. For example, 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'.

3. There is a *postprocessing issue*: although A' is "correct" in some technical sense, it need not necessarily produce the same outputs as an ideal algorithm A^*. For example, suppose A computes the Voronoi diagram of a set of points in the plane. Four cocircular points are a degeneracy and are 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

We have two general methods for converting a generic algorithm to a general one:

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 evaluations of test values by calls to this blackbox. In [Yap90b], a family of ***admissible schemes*** for blackboxes is 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 nondegenerate input $\mathbf{a}^*$ for the problem, then every other input $\mathbf{a}$ can be made nondegenerate by perturbing it in the direction of $\mathbf{a}^*$. We can take the perturbed input to be $\mathbf{a} + \epsilon\mathbf{a}^*$ 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 $(t, t^2, \ldots, t^n)$.

We compare these two approaches. We currently only have blackbox schemes for rational functions, while Seidel's method would apply even in nonalgebraic settings. Blackbox schemes are independent of particular problems, while the nondegenerate instances $\mathbf{a}^*$ depend on the problem (and on the input size); no systematic method to choose $\mathbf{a}^*$ is known.

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 (for determinants) turns out to be an admissible scheme [Yap90b]. Intuitively, sign blackbox invocations should be almost as fast as the actual evaluations with high probability [Yap90b]. But the worst-case exponential behavior led Emiris and Canny to propose more efficient numerical approaches [EC95]. To each input parameter a_i in $\mathbf{a}$, they add a perturbation $b_i\epsilon$ (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 the b_i's will ensure nondegeneracy and efficient computation. For general rational function tests, a lemma of Schwartz show that a random choice of the b_i's is likely to yield nondegeneracy. Emiris, Canny, and Seidel [ECS94, Sei94] give a general result on the validity of linear perturbations, and apply it to common test polynomials.

APPLICATIONS AND PRACTICE

Michelucci [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 are reported in [ECS94]. Neuhauser [Neu97] improved and implemented the rational blackbox scheme of Yap. He also considered controlled perturbation techniques. Comes and Ziegelmann [CZ99] implemented the linear perturbation ideas of Seidel in CGAL.

In solid modeling systems, it is very useful to systematically avoid degenerate cases (numerous in this setting). Fortune [For97] uses symbolic perturbation to allow an "exact manifold representation" of nonregularized polyhedral solids (see Section 47.1). The idea is that a 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.

In contrast to our general goal of eliminating *explicit* handling of degeneracies, there are a few papers on "perturbation" that proposes to directly handle degeneracies. Burnikel, Mehlhorn, and Schirra [BMS95] describe the implementation of a line segment intersection algorithm and semi-dynamic convex hull maintenance in arbitrary dimensions. Based on 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 postprocessing problem can be significant; and (iii) it is not hard to handle degeneracies directly. But the probability of (i) occurring in a drastic way (e.g., for a degenerate input of n identical points) is so negligible that it may not deter most users when they have the option of writing a generic algorithm, especially when the general algorithm is very complex or not readily available. Other experiences suggest that property (iii) is the exception rather than the rule. In any case, users must weigh these considerations (Cf. [Sch94]).

A weaker form of the [BMS95] approach is illustrated by work of Halperin and co-workers [HS98, Raa99]. Again, the algorithm must explicitly detect the presence of degeneracies but now, we explicitly perturb the input to remove all degeneracies. Their problem may be framed as follows: given a sequence $S = (O_1, \ldots, O_n)$ of geometric objects, let A_i $(i = 1, \ldots, n)$ be the arrangement formed by $S_i = (O_1, \ldots, O_i)$. The goal is to compute $A_n = A(S_n)$. For any object O and $\varepsilon > 0$, consider a predicate $P_1(O, \varepsilon)$ with this ***monotonicity property***: if $\varepsilon' > \varepsilon$ and $P_1(O, \varepsilon')$ is true then $P_1(O, \varepsilon)$ is true. Call P_1 an ***approximate degeneracy predicate***. If $P_1(O, \varepsilon)$ is true, we say O is ε***-degenerate***. Also, $P_1(O, 0^+)$ reduces to standard notions of degeneracy. Such predicates may be defined by a Boolean combination of polynomial inequalities. For instance, let O be a curve and $P_1(O, \varepsilon)$ is true iff there is a δ-ball B centered at a point of O, $\delta \leq \varepsilon$, such that $B \cap O$ is not connected. Thus $P_1(O, 0^+)$ is the property that O is self-intersecting. In general, let P_k denote an approximate degenerate predicate on $k \geq 1$ distinct objects. If P_k and P'_k are two such predicates, then so is $P_k \vee P'_k$ and $P_k \wedge P'_k$. For instance, $P_2(O_1, O_1, \varepsilon)$ might say that O_1, O_2 are ε-close. Fix a collection $\mathcal{P}$ of approximate degeneracy predicates. We say that S is ε***-degenerate*** if for some $P_k \in \mathcal{P}$, $P_k(O_1, \ldots, O_k, \varepsilon)$ is true for some choice of k distinct objects $O_1, \ldots, O_k \in S$. The following ε-δ ***perturbation estimation problem*** is basic: given $\varepsilon > 0$, find $\delta = \delta(\varepsilon, S, O) > 0$

such that if S is non ε-degenerate, and O is any object, with probability $> 1/2$, a random δ-perturbation O' of O will form a non ε-degenerate configuration with S. By general principles, we know that δ exists; but we would like good bounds on δ (say polynomial in $|S|$, etc). Using this, we can solve the ***perturbed arrangement problem***: given S and $\varepsilon > 0$, compute an arrangement $A(S')$ where S' is not ε-degenerate and S' is a δ-perturbation of S. The cited papers above solve the perturbed arrangement problem in two situations, when the objects are spheres and polyhedral surfaces, respectively. The idea is to use a form of randomized incremental construction.

41.6 OPEN PROBLEMS

1. The main theoretical question in EGC is whether the Constant Zero Problem for Ω_4 is decidable. A related, possibly simpler, question is whether $\mathrm{ZERO}(\Omega_3 \cup \{\sin(\cdot), \pi\})$ is decidable.

2. In constructive root bounds, it is unknown if there exists a root bound $\beta : \mathcal{E}(\Omega_2) \to \mathbb{R}_{\geq 0}$ where $-\lg(\beta(E)) = O(D(E))$ and $D(E)$ is the degree of E. In current bounds, we only know a quadratic bound, $-\lg(\beta(E)) = O(D(E)^2)$. The Uniformity Conjecture of Richardson [Ric00], if true, would be a very deep result with practical applications.

3. Give a optimal algorithm for the guaranteed precision evaluation problem $\mathrm{GVAL}(\Omega)$ for, say, $\Omega = \Omega_2$. The solution includes a reasonable cost model.

4. In geometric rounding, we pose two problems: (a) Extend the Greene-Yao rounding problem to non-uniform grids (e.g., the grid points are L-bit floating point numbers). (b) Round simplicial complexes. The preferred notion of rounding here should not increase combinatorial complexity (unlike Greene-Yao), allow features to collapse (triangles can degenerate to a vertex), but disallow inversion (triangles cannot flip its orientation).

5. Give good bounds for the ε-δ perturbation estimation problem.

6. Give a systematic treatment of inexact (dirty) data. Held [Hel01a, Hel01b] describes the engineering of reliable algorithms to handle such inputs.

41.7 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). Other surveys of robustness issues in geometric computation are Schirra [Sch99], Yap and Dubé [YD95] and Fortune [For93]. Robust geometric modelers are surveyed in [PCH+95].

RELATED CHAPTERS

Chapter 21: Arrangements
Chapter 24: Shortest paths and networks
Chapter 29: Computational real algebraic geometry
Chapter 47: Solid modeling
Chapter 52: Computational geometry software

REFERENCES

[AKY02] Tetsuo Asano, David Kirkpatrick, and Chee Yap. Pseudo approximation algorithms, with applications to optimal motion planning. In *ACM Symp. on Computational Geometry*, volume 18, pages 170–178, 2002. Barcelona, Spain. To appear, Special Conference Issue of J.Discrete & Comp. Geom.

[BBP01] Hervé Brönnimann, Christoph Burnikel, and Sylvain Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1-2):25–47, 2001. Also: Proc. 14th ACM Symp. Comput. Geom. (1998).

[BCD+02] G. Baraquet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002.

[BEH+02] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *Proc. ESA 2002*, 2002. To Appear.

[BFM+01] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In *Lecture Notes in Computer Science*, pages 254–265, 2001.

[BFMS99] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Exact geometric computation made easy. In *Proc. 15th ACM Symp. Comp. Geom.*, pages 341–450, 1999.

[BFMS00] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27:87–99, 2000.

[BFS98] C. Burnikel, S. Funke, and M. Seel. Exact geometric predicates using cascaded computation. In *Proc. 14th Annual Symp. Computational Geometry*, pages 175–183, 1998.

[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.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. of the ACM*, 42(1):269–291, January 1995.

[BKM+95] Christoph Burnikel, Jochen Könnemann, Kurt Mehlhorn, Stefan Näher, Stefan

Schirra, and Christian Uhrig. Exact geometric computation in LEDA. In *Proc. 11th ACM Symp. Comp. Geom.*, pages C18–C19, 1995.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing and self-correcting programs, with applications to numerical programs. *J. of Computer and System Sciences*, 47:549–595, 1993.

[BMS94] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In *Lecture Notes in Computer Science*, volume 855. Springer, 1994. Proceedings of ESA'94.

[BMS95] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 16–23, 1995.

[BS00] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. *Computational Geometry: Theory and Applications*, 16(1), 2000.

[Bur96] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections.* Ph.D thesis, Universität des Saarlandes, March 1996.

[BY00] H. Brönnimann and M. Yvinec. Efficient exact evaluation of signs of determinants. *Algorithmica*, 27:21–56, 2000.

[Can88] John Francis Canny. *The complexity of robot motion planning.* ACM Doctoral Dissertion Award Series. The MIT Press, Cambridge, MA, 1988. PhD 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.

[Cho99] Timothy Y. Chow. What is a closed-form number? *Amer. Math. Monthly*, 106(5):440–448, 1999.

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

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

[CSY97] J. Choi, J. Sellen, and C. Yap. Approximate Euclidean shortest path in 3-space. *Int'l. J. Computational Geometry and Applications*, 7(4):271–295, 1997. Also: Proc. 10th ACM Symp. on Comp. Geom., p.41–48, 1994.

[CZ99] Jochen Comes and Mark Ziegelmann. An easy to use implementation of linear perturbations within cgal. In *Proc. 3rd Workshop on Algorithm Engineering (WAE99)*, Berlin, 1999. LNCS 1668 Springer.

[DEMY02] Z. Du, M. Eleftheriou, J. Moreira, and C. Yap. Hypergeometric functions in exact geometric computation. In V.Brattka, M.Schoeder, and K.Weihrauch, editors, *Proc. 5th Workshop on Computability and Complexity in Analysis*, pages 55–66, 2002. Malaga, Spain, July 12-13, 2002. In Electronic Notes in Theoretical Computer Science, 66:1 (2002), `http://www.elsevier.nl/locate/entcs/volume66.html`. Also available as "Computability and Complexity in Analysis", Informatik Berichte No.294-6/2002, Fern University, Hagen, Germany.

[DP99] Olivier Devillers and Franco P. Preparata. Further results on arithmetic filters for geometric predicates. *Computational Geometry: Theory and Applications*, 13(2):141–148, 1999.

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

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

[EC95] I. Z. Emiris and J. F. Canny. A general approach to removing degeneracies. *SIAM J. 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] Steven J. Fortune and Victor J. Milenkovic. Numerical stability of algorithms for line arrangements. *ACM Symp. on Computational Geometry*, 7:334–341, 1991.

[FMN99] Stefan Funke, Kurt Mehlhorn, and Stefan Näher. Structural filtering: A paradigm for efficient and exact geometric programs. In *Proc. 11th Canadian Conference on Computational Geometry*, 1999.

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

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

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

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

[For97] Steven J. Fortune. Polyhedral modeling with multiprecision integer arithmetic. *Computer-Aided Design*, pages 123–133, 1997. Also, 3rd ACM SIGGRAPH Symp. Solid Modeling and Appl. (1995).

[For98] Steven Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. In *Proc. ACM Symp. on Comp. Geometry*, 1998.

[Fun97] Stefan Funke. Exact arithmetic using cascaded computation. Master's thesis, Max Planck Institute for Computer Science, Saarbrücken, Germany, 1997.

[FvW93a] Steven J. Fortune and Christopher J. van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th ACM Symp. on Computational Geom.*, pages 163–172, 1993.

[FvW93b] Steven J. Fortune and Christopher J. van Wyk. LN User Manual, 1993. AT&T Bell Laboratories.

[FvW96] Steven J. Fortune and Christopher J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15(3):223–248, 1996.

[Gae99] Bernd Gaertner. Exact arithmetic at low cost - a case study in linear programming. *Computational Geometry: Theory and Applications*, 13(2):121–139, 1999.

[GGHT97] M. Goodrich, L. J. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 284–293, 1997.

[GHS01] Nicola Geismann, Michael Hemmer, and Elmar Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th ACM Symp. on Computational Geometry*, 2001.

[GM95] Leo Guibas and D. Marimont. Rounding arrangements dynamically. In *Proc. 11th ACM Symp. Computational Geom.*, pages 190–199, 1995.

[GS00] Bernd Gaertner and Sven Schoenherr. An efficient, exact, and generic quadratic programming solver for geometric optimization. *ACM Symp. on Computational Geometry*, 16:??, 2000.

[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.

[Hel01a] Martin Held. FIST: Fast industrial-strength triangulation of polygons. *Algorithmica*, 30(4):563–596, 2001.

[Hel01b] Martin Held. VRONI: An engineering appraoch to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry: Theory and Applications*, 18:95–123, 2001.

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

[HHK+01] Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Michael Seel. An adaptable and extensible geometry Kernel. In *Proc. 5th Int'l Workshop on Algorithm Engineering (WAE-01)*, pages 79–90, Berlin, 2001. Springer. Aarhus, Denmark, August 28 - 30, 2001.

[Hob99] John D. Hobby. Practical segment intersection with finite precision output. *Computational Geometry: Theory and Applications*, 13:199–214, 1999.

[HS98] D. Halperin and C.R. Shelton. A perturbation scheme for spherical arrangements with applications to molecular modeling. *Computational Geometry: Theory and Applications*, 10(4):273–288, 1998.

[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.

[KFC+01] Shankar Krishnan, Mark Foskey, Tim Culver, John Keyser, and Dinesh Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. *ACM Symp. on Computational Geometry*, 17:274–283, 2001.

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

[KLPY99] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A Core Library for robust numerical and geometric libraries. In *15th ACM Symp. Computational Geometry*, pages 351–359, 1999.

[KW98] Lutz Kettner and Emo Welzl. One sided error predicates in geometric computing. In Kurt Mehlhorn, editor, *Proc. 15th IFIP World Computer Congress, Fundamentals - Foundations of Computer Science*, pages 13–26, 1998.

[LPT97] G. Liotta, F. Preparata, and R. Tamassia. Robust proximity queries: an illustration of degree-driven algorithm design. *ACM Symp. on Computational Geometry*, 13:156–165, 1997.

[LY01] Chen Li and Chee Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505. ACM and SIAM, January 2001.

[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.

[Mig82] Maurice Mignotte. Identification of algebraic numbers. *J. of Algorithms*, 3:197–204, 1982.

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

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

[MN95] Kurt Mehlhorn and Stefan Näher. LEDA: a platform for combinatorial and geometric computing. *CACM*, 38:96–102, 1995.

[MNS+96] K. Mehlhorn, S. Näher, T. Schilz, R. Seidel, M. Seel, and C. Uhrig. Checking geometric programs or verification of geometric structures. In *Proc. 12th ACM Symp. on Computational Geom.*, pages 159–165. Association for Computing Machinery, May 1996.

[MW96] Angus Macintyre and A. Wilkie. On the decidability of the real exponential field. In *Kreiseliana, About and Around Georg Kreisel*, pages 441–467. A.K. Peters, 1996.

[Neu97] Michael A. Neuhauser. Symbolic perturbation and special arithmetics for controlled handling of geometric degeneracies. In *Proc. 5th Int'l. Conf. in Central Europe on Computer Graphics and Visualization (WSCG'97)*, pages 386–395, 1997. Plzeň, Czech Republic.

[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 modelers, 1994 progress report. In *Proc. 1995 NSF Design and Manufacturing Grantees Conference*, pages 139–140, 1995.

[PY01] Victor Y. Pan and Yanqiang Yu. Certification of numerical computation of the sign of the determinant of a matrix. *Algorithmica*, pages 708–724, 2001.

[PY03] Sylvain Pion and Chee Yap. Constructive root bound method for k-ary rational input numbers. In *Proc. 19th ACM Symp. on Comp. Geom.*, page (To Appear), 2003. San Diego, California.

[Raa99] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Symposium on Computational Geometry*, pages 163–172, 1999.

[Ric97] Daniel Richardson. How to recognize zero. *J. of Symbolic Computation*, 24:627–645, 1997.

[Ric00] Daniel Richardson. The uniformity conjecture. In J. Blank, V. Brattka, and P. Hertling, editors, *Computability and Complexity in Analysis*. Springer, 2000. 4th International Workshop, CCA 2000, Swansea, UK, September 17-19, 2000, Selected Papers, Lecture Notes in Computer Science, No. 2064.

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

[Sch99] Stefan Schirra. Robustness and precision issues in geometric computation. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science Publishers, B.V. North-Holland, Amsterdam, 1999.

[Sch00] Edward R. Scheinerman. When close enough is close enough. *Amer. Math. Monthly*, 107:489–499, 2000.

[SCY00] J. Sellen, J. Choi, and C. Yap. Precision-sensitive Euclidean shortest path in 3-Space. *SIAM J. Computing*, 29(5):1577–1595, 2000. Also: 11th ACM Symp. on Comp. Geom., (1995)350–359.

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

[See01] Michael Seel. *Planar Nef Polyhedra and Generic High-dimensional Geometry*. Phd thesis, Universitt des Saarlandes, 2001.

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

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

[She96] Jonathan Richard Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th ACM Symp. on Computational Geom.*, pages 141–150. Association for Computing Machinery, May 1996.

[SI89a] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. *J.Information Processing, Information Processing Society of Japan*, 12(4):380–393, 1989.

[SI89b] K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite precision arithmetic. *Applied Mathematics Letters*, 2:203–206, 1989.

[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.

[SIII00] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation – an approach to robust geometric algorithms. *Algorithmica*, 27, 2000.

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

[SSTY00] E. Schömer, J. Sellen, M. Teichmann, and C. Yap. Smallest enclosing cylinders. *Algorithmica* 27, 2000, 170–186.

[Sug89] Kokichi Sugihara. On finite-precision representations of geometric objects. *J. of Computer and System Sciences*, 39:236–247, 1989.

[Ull90] Christian Ullrich, editor. *Computer Arithmetic and self-validating numerical methods*. Academic Press, Boston, 1990.

[Wei02] Ron Wein. High level filtering for arrangements of conic arcs. In *Lecture Notes in Computer Sci., vol. 2461*, pages 884–895. Springer-Verlag, 2002. Proc. 10th European Symposium on Algorithms (ESA 2002), Rome.

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

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

[Yap97] Chee K. Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7:3–23, 1997. Invited talk, Proceed. 5th Canadian Conference on Comp. Geometry, Waterloo, Aug 5–9, 1993.

[Yap98] Chee Yap. A new number core for robust numerical and geometric libraries. In *3rd CGC Workshop on Geometric Computing*, 1998. Invited Talk. Brown University, Oct 11–12, 1998. For abstracts, see `http://www.cs.brown.edu/cgc/cgc98/home.html`.

[Yap00] Chee K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000. A version is available at URL `ftp:/Preliminary/cs.nyu.edu/pub/local/yap/algebra-bk`.

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

[Yu92] Jiaxun Yu. *Exact arithmetic solid modeling*. Ph.D. dissertation, Department of Computer Science, Purdue University, West Lafayette, IN 47907, June 1992. Technical Report No. CSD-TR-92-037.