45 ROBUST GEOMETRIC COMPUTATION

Vikram Sharma and Chee K. Yap

INTRODUCTION

Nonrobustness refers to qualitative or catastrophic failures in geometric algorithms arising from numerical errors. Section 45.1 provides background on these problems. Although nonrobustness is already an issue in "purely numerical" computation, the problem is compounded in "geometric computation." In Section 45.2 we characterize such computations. Researchers trying to create robust geometric software have tried two approaches: making fixed-precision computation robust (Section 45.3), and making the exact approach viable (Section 45.4). Another source of nonrobustness is the phenomenon of degenerate inputs. General methods for treating degenerate inputs are described in Section 45.5. For some problems the exact approach may be expensive or infeasible. To ensure robustness in this setting, a recent extension of exact computation, the so-called "soft exact approach," has been proposed. This is described in Section 45.6.

45.1 NUMERICAL NONROBUSTNESS ISSUES

Numerical nonrobustness in scientific computing is a well-known and widespread phenomenon. The root cause is the use of ***fixed-precision numbers*** 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 1980s 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.

Nonrobustness is already problematic in purely numerical computation: this is well documented in numerous papers in numerical analysis with the key word "pitfalls" in their title. But nonrobustness apparently becomes intractable in "geometric" computation. In Section 45.2, 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, some suitable *fixed-precision geometry* is needed to substitute for the original geometry (which is usually Euclidean). We describe such approaches in Section 45.3. But in Section 45.4, we describe the alternative *exact approach* that requires arbitrary precision. Naively, the exact approach suggests that each numerical predicate must be computed exactly. But the formulation of ***Exact Geometric Computation*** (EGC) asks only for error-free evaluation of predicates. The simplicity and generality of the EGC solution has made it the dominant nonrobustness approach among computational geometers.

In Section 45.5, we address a different but common cause of numerical nonrobustness, namely, *data degeneracy*. Geometric inputs can be degenerate: an input triangle might degenerate into a line segment, or an input set of points might contain collinear triples, etc. This can cause geometric algorithms to fail. But if geometric algorithms must detect such special situations, the number of such special cases can be formidable, especially in higher dimensions. For nonlinear problems, such analysis usually requires new and nontrivial facts of algebraic geometry. This section looks at general techniques to avoid explicit enumeration of degeneracies.

In Section 45.6, we note some formidable barriers to extending the EGC approach for nonlinear and nonalgebraic problems. This motivates some current research directions that might be described as *soft exact computation*. It goes beyond "simply exact" computation by giving a formal role to numerical approximations in our computing concepts and notions of correctness.

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. The representation of floating point numbers using these L bits is dictated by the IEEE Floating Point standard. ***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: They refer to software packages for representing and computing with arbitrary precision numbers. There are three main types of such number packages, called *BigIntegers*, *BigRationals* and *BigFloats*, representing (respectively) integers, rational numbers, and floating point numbers. For instance, $+$, $-$, and $\times$ are implemented exactly with *BigIntegers*. With *BigRationals*, division can also be exact. Beyond rational numbers, *BigFloats* become essential, and operations such as $\sqrt{\ }$ can be approximated to any desired precision in *BigFloat*. But ensuring that *BigFloats* achieve the correct rounding is a highly nontrivial issue especially for transcendental function. The `MPFR` [FHL+07] package is the only Big number package to address this.

45.2 THE NATURE OF GEOMETRIC COMPUTATION

It is well known that numerical approximations may cause an algorithm to crash or enter an infinite loop; but there is a persistent belief that when the algorithm halts, then the output is a reasonably close approximation (up to the machine precision) to the true output. The paper [KMP$^+$07, §4.3] wishes to "refute this myth" by considering a simple algorithm for computing the convex hull of a planar point set. They constructed inputs such that the output "convex hull" might (1) miss a point far from the interior, (2) contain a large concave corner, or (3) be a self-intersecting polygon. The paper provides "classroom examples" based on a systematic analysis of floating point errors and their effects on common predicates in computational geometry. If the root cause of 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 a 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 is on a line," "a point is inside a simplex," or "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 31) is geometric. See Table 45.2.1 for further examples from well-known problems.

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 28), with suitable additional labels and constraints, is a primary example of such a structure. The reader may keep this example in mind in the following definition. A ***geometric structure***

TABLE 45.2.1 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

is comprised of four components:

$$D = (G, \lambda, \Phi(\mathbf{z}), I), \tag{45.2.1}$$

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***, for an input of size n, is $I : \{z_1, \ldots, z_n\} \to \mathbb{R}$ where the z_i's are called ***structural variables***. We informally identify I with the sequence "$\mathbf{c} = (c_1, \ldots, c_n)$" where $I(z_i) = c_i$. The c_i's are called ***(structural) parameters***. For each $u \in V \cup E$, the label $\lambda(u)$ is a Tarski formula of the form $\xi(\mathbf{x}, \mathbf{z})$, where $\mathbf{z} = (z_1, \ldots, z_n)$ are the structural variables and $\mathbf{x} = (x_1, \ldots, x_d)$ for some $d \geq 1$. This d is fixed and determines the ambient space $\mathbb{R}^d$ containing the geometric object. This formula defines a ***semialgebraic set*** (Chapter 37) parameterized by the structural variables. For a given $\mathbf{c}$, the semialgebraic set is $f_{\mathbf{c}}(v) = \{\mathbf{a} \in \mathbb{R}^d \mid \xi(\mathbf{a}, \mathbf{c}) \text{ holds}\}$. Following Tarski, we are identifying semialgebraic sets in $\mathbb{R}^d$ with d-dimensional geometric objects. The consistency relation $\Phi(\mathbf{z})$ is another Tarski formula of the form $\Phi(\mathbf{z}) = (\forall x_1, \ldots, x_d)\phi(\lambda(u_1), \ldots, \lambda(u_m), \mathbf{x}, \mathbf{z})$ where $u_1, \ldots, u_m$ ranges over elements of $V \cup E$. For each class of geometric structures, e.g., hyperplane arrangements, or Voronoi diagrams of points, the formula $\Phi(\mathbf{z})$ can be systematically constructed from G. Note that if D is to be computed in an output, we need not explicitly specify $\Phi(\mathbf{z})$, as this is understood (or implicit in our understanding of the geometric structure). The definition above can be contrasted with Fortune's definition, where a geometric problem on input of size n is a map from $\mathbb{R}^n$ to a discrete set, e.g., the set of cyclic permutations for convex hulls or the incidence graph for arrangements [For89]; this definition, however, fails to capture discrete relations on the input.

An example of the notation in (45.2.1), is 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 $\mathrm{aff}(X)$ denote the affine span of a set $X \subseteq \mathbb{R}^d$. 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 halfspaces defined by $\mathrm{aff}(f(u)) \setminus \mathrm{aff}(f(v))$. We let $\lambda(u,v)$ be the Tarski formula $\xi(\mathbf{x},\mathbf{z})$ that defines the open halfspace in $\mathrm{aff}(f(u))$ that contains $f(u)$. Again, 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 appears complicated, all 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

$$\widehat{D} = (G, \lambda, \Phi(\mathbf{z})) \tag{45.2.2}$$

becomes an ***abstract geometric structure***, and $D = (G, \lambda, \Phi(\mathbf{z}), I)$ is an ***instance*** of $\widehat{D}$. The structure D in (45.2.1) 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 the Euclidean norm of $\mathbf{c} - \mathbf{c}'$.

The graph $G = (V,E)$ in (45.2.1) is an abstract graph where each $v \in V$ is a symbol that represents the semi-algebraic set $f(v) \subseteq \mathbb{R}^d$. The Tarski formula $\lambda(v)$ is an exact but symbolic representation of $f(v)$. For most applications of geometric algorithms, such a symbolic representation is alone insufficient. We need an approximate "embedding" of the underlying semi-algebraic sets in $\mathbb{R}^d$. Consider the problem of meshing curves and surfaces (see Boissonnat et al. [BCSM$^+$06] for a survey). In meshing, the set $\{f(v) : v \in V\}$ is typically a simplicial complex (i.e., triangulation). For each $v \in V$, if $f(v)$ is a k-simplex ($k = 0, \ldots, d$), then we want to compute a piecewise linear set $\widetilde{f}(v) \subseteq \mathbb{R}^d$ that is homeomorphic to a k-ball. Moreover, $(u,v) \in E$ iff $\widetilde{f}(v) \subseteq \partial \widetilde{f}(u)$ (∂ is the boundary operator). Then the set $\widetilde{V} = \{\widetilde{f}(v) : v \in V\}$ is a topological simplicial complex that captures all information in the symbolic graph $G = (V,E)$. The algorithms in [PV04, LY11, LYY12] (for $d \leq 3$) can even ensure that for all $v \in V$, we have $d_H(f(v), \widetilde{f}(v)) < \varepsilon$ (for any given ε) where d_H denotes the Hausdorff distance on Euclidean sets. In that case, $\widetilde{V}$ is an ***ε-approximation*** of the simplicial complex $\{f(v) : v \in V\}$. This is the "explicit" geometric representation of D, which applications need.

45.3 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 45.3.1 and in Table 45.3.1.

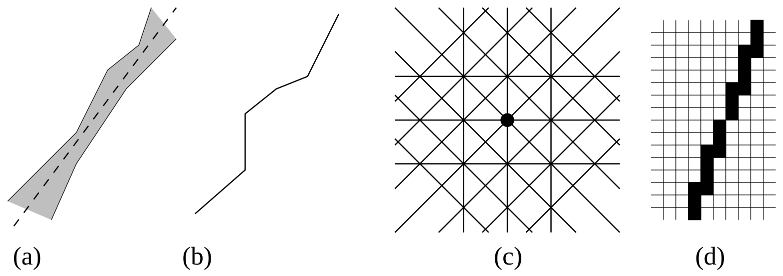

FIGURE 45.3.1
Four concepts of "finite-precision" lines.

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 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 45.3.1 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, Mil89]
(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 45.3.1(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] advocate 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 constructs geometric algorithms that preserve the chosen property in the following sense: the algorithm will always terminate with an output whose topological properties match the topological properties for the output corresponding to some input but not necessarily a nearby instance. It is not clear in this prescription how to choose appropriate topological properties. Greene and Yao [GY86] 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 [For99] in 3 dimensions. 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 45.2) 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 structures (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. A similar approach has been proposed more recently by Sugihara [Sug11].

STABILITY

This is a metric form of topological distortion where we place a priori bounds on the amount of distortion. It is analogous to backward 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, following Fortune [For89], that 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, For95]. 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 in [SI92] have not been proved 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 are polylines 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 45.5, 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.

45.4 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 a 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 45.2) 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 $c_1, \ldots, c_n$ (this is I in (45.2.1)). A nonbranching node at depth i computes a value v_i, say $v_i \leftarrow f_i(v_1, \ldots, v_{i-1}, c_1, \ldots, c_n)$. 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 take 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 $|z - \widetilde{z}| \le \max\{2^{-a}, |z|2^{-r}\}$. When $r = \infty$, then $z[\infty, a]$ comprises all the numbers $\widetilde{z}$ that approximate 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 approximate 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, ZERO(Ω)***:*** Given $E \in \mathcal{E}(\Omega)$, decide if $\mathrm{val}(E) = \uparrow$; if not, decide if $\mathrm{val}(E) = 0$.

Guaranteed Precision Evaluation Problem, GVAL(Ω)***:*** 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 [FW93a, 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 number 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 45.4.1.

One of `LN`'s goals [FW96] 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 [BJMM93] is to delay evaluating an expression until forced to, and to maintain intervals of uncertainty for values. Upon complete evaluation, the expression is discarded. It uses root bounds to achieve exactness

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

TABLE 45.4.1 Expression packages.

SYSTEM	DESCRIPTION	REFERENCES
LN	Little Numbers	[FW96]
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, YYD+10]

and floating point filters for speed. The Real/Expr Package [YD95] was the first package to achieve guaranteed precision for a general class of nonrational expressions. It introduces the "precision-driven mechanism" whereby a user-specified precision at the root of the expression is transformed and downward-propagated toward the leaves, while approximate values generated at the leaves are evaluated and error bounds up-propagated up to the root. This up-down process may be iterated. See [LPY04] for a general description of this evaluation mechanism, and [MS15] for optimizations issues. LEDA Real [BFMS99, BKM+95] 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 68). The semantics of Real/Expr 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 are the adoption of conventional assignment semantics and the introduction of a simple ***Numerical Accuracy API*** [Yap98].

The CGAL Library (Chapter 68) 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 a solution to the decision problem for Tarski's language, on the associated cell decomposition problems, as well as cell adjacency computation (Chapter 37). However, general EGC libraries such as the 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***. The first is the Constant Zero Problem ZERO(Ω). But there are two closely related problems. In the ***Constant Va-***

lidity Problem VALID(Ω), we are to decide if a given $E \in \mathcal{E}(\Omega)$ is valid, i.e., $\mathrm{val}(E) \neq \uparrow$. The ***Constant Sign Problem*** SIGN(Ω) is to compute $\mathtt{sign}(E)$ for any given $E \in \mathcal{E}(\Omega)$, where $\mathtt{sign}(E) \in \{\uparrow, -1, 0, +1\}$. In case $\mathrm{val}(E)$ is complex, define $\mathtt{sign}(E)$ to be the sign of the real part of $\mathrm{val}(E)$.

TABLE 45.4.2 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	$\Omega_2^+ = \Omega_2 \cup \{\sqrt[k]{\cdot} : k \geq 3\}$
$\Omega_3 = \Omega_2 \cup \{\mathrm{RootOf}(P(X), I)\}$	Algebraic Numbers	Use of $\diamond(E_1, \ldots, E_d, i)$, [BFM+09]
$\Omega_4 = \Omega_3 \cup \{\exp(\cdot), \ln(\cdot)\}$	Elementary Numbers (cf. [Cho99])	

There is a natural hierarchy of the expression classes, each corresponding to a class of complex numbers as shown in Table 45.4.2. 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 a 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 $\mathrm{RootOf}(P, I)$ can be generalized to allow expressions as coefficients of $P(X)$ as in Burnikel et al. [BFM+09], or by introducing systems of polynomial equations as in Richardson [Ric97]. The problem of isolating the roots of such a polynomial in the univariate case has been recently addressed in [BSS+16, BSSY15]. Although Ω_4 can be treated as a set of real operators, it is more natural to 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 that ZERO(Ω) and VALID(Ω) are reducible to SIGN(Ω). For Ω_4, all three problems are recursively equivalent. The fundamental problems related to Ω_i are 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 intriguing conditional result of Richardson [Ric07] is that ZERO(Ω_4) is decidable, conditional on the truth of Schanuel's conjecture in Transcendental Number Theory. The most general unconditional result known about ZERO(Ω_4) is (essentially) Baker's theory of linear form in logarithms [Bak75].

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. A ***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 $\mathrm{val}(E) \neq 0$ then $|\mathrm{val}(E)| \geq \beta(E)$. More precisely, β is called an ***exclusion*** root bound; it is an ***inclusion root bound*** when the inequality becomes "$|\mathrm{val}(E)| \leq \beta(E)$." 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 $\mathrm{val}(E)$ such that $|\alpha - \mathrm{val}(E)| < \beta(E)/2$. 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 $\mathrm{val}(E) = 0$ if $|\alpha| \leq \beta/2$. Otherwise $|\alpha| > \beta/2$, and the sign of $\mathrm{val}(E)$ is that of α. An important remark is that the root bound β determines the worst-case complexity. This is unavoidable if $\mathrm{val}(E) = 0$. But if $\mathrm{val}(E) \neq 0$, 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 $\mathtt{sign}(\mathrm{val}(E)) = \mathtt{sign}(\alpha_i) \neq 0$.

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$^+$09] 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. Scheinerman [Sch00] provides an interesting bound based on eigenvalues. A factoring technique of Pion and Yap [PY06] can be combined with the above methods (e.g., BFMSS) to yield sharper bounds; it exploits the presence of k-ary input numbers (such as binary or decimal numbers) which appear in realistic inputs. In general, we need multivariate root bounds such as ***Canny's bound*** [Can88], the ***Brownawell-Yap bound*** [BY09], and the ***DMM bound*** [EMT10].

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 $|\mathrm{val}(E) - \alpha| \leq \varepsilon$. 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 [FW96]. Floating-point filters can be classified along the static-to-dynamic dimension: ***static filters*** compute the bound ε solely from information that is 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 [FW96]) 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 [FW93b, Fun97]. Filters can be 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 [FMN05]. 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 A be a (standard) ***algorithm*** for P which, viewed as a total function $A : I \to O \cup \{\uparrow\}$, has the property that for all $i \in I$, we have $A(i) \neq \uparrow$ iff $(i, A(i)) \in P$. Let $H : I \to O \cup \{\uparrow\}$ be another algorithm with no restrictions; call H a ***heuristic algorithm*** for P. Let $F : I \times O \to \{true, false\}$. Then F is a ***checker*** for P if F computes the characteristic function for P, i.e., $F(i, o) = true$ iff $(i, o) \in P$. Note that F is a checker for the problem P, and not for any purported program for P. Unlike program checking literature, 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. Another filter F' is ***more efficacious*** than F if $F(i, o) = true$ implies $F'(i, o) = true$. A ***filtered program*** for P is therefore a triple (H, F, A) where H is a 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. In illustration, let P be the problem $SDET$ of computing the sign of determinants. The only checker we know here is trivial, amounting to computing the determinant itself. On the other hand, effective filters for $SDET$ are known [BBP01, PY01].

PRECISION COMPLEXITY

An important goal of EGC is to control the cost of high-precision computation. For instance, consider the sum of square roots problem: given $2k$ numbers $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ such that $0 < a_i, b_i \leq N$, derive a bound on the precision required to compute a 1-relative bit approximation to $(\sum_{i=1}^k \sqrt{a_i} - \sum_{i=1}^k \sqrt{b_i})$. The best known bounds on the precision are of the form $O(2^k \log N)$; for instance, we can compute the constructive root bound for the expression denoting the difference and take its logarithm to obtain such a bound. In practice, it has been observed that $O(k \log N)$ precision is sufficient. The current best upper bound implies that the problem is in PSPACE. We next describe two approaches to address the issue of precision complexity 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 31.5), 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. The first such scheme from Papadimitriou [Pap85] 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 45.2) 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. [AKY04] gave new and more efficient 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 45.3) to robustness. But geometric rounding is also important in EGC, with a difference. The EGC problem is to "round" a geometric structure (Section 45.2) 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 connects to a larger literature on simplification of geometry (e.g., simplifying geometric models in computer graphics and visualization (Chapter 52). Two distinct 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 by 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). In `Core Library` 2.1, transcendental functions are provided, but they are computed up to some user-specified "Escape Bound." Numbers that are smaller than this bound are declared zero, and a record is made. Thus our computation is correct subject to these records being actual zeros. Another intriguing approach is to invoke some strong version of the uniformity conjecture [RES06], which provides a bound implemented in an EGC library like [YYD+10], or to use a decision procedure conditioned on the truth of Schanuel's conjecture [Ric07]. If our library ever led to an error, we would have produced a counterexample to the underlying conjectures. Thus, we are continually testing some highly nontrivial conjectures when we use the transcendental parts of the library.

A rare example of a simple geometric problem that is provably transcendental and yet decidable was shown in [CCK+06]. This is the problem of shortest path between two points amidst a set of circular obstacles. Although a direct argument can be used to show the decidability of this problem, to get a complexity bound, it was necessary to invoke Baker's theory of linear form in logarithms [Bak75] to derive a single-exponential time bound.

The need for transcendental functions may be only apparent: e.g., path planning in Euclidean space only appears to need trigonometric functions, but they can be replaced by algebraic relations. Moreover, we can get arbitrarily good approximations by using *rational rigid transformations* where the sines or cosines are rational. 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 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., [FW96]). 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 [LPT98, 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***; see Chapter 57. 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 the 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 is to produce such solutions. **(iv)** An area of major challenge is computation of algebraic curves and surfaces. Algorithms for low degree curves and surfaces can be efficiently solved today e.g., [BEH$^+$02, GHS01, Wei02, EKSW06]. Arbitrary precision libraries for algebraic curves have been implemented (see Krishnan et al. [KFC$^+$01]) but without zero bounds, there are no topology guarantees. More recently, there has been a lot of progress in determining the topology of curves, which may be given implicitly, e.g., as the Voronoi diagram of ellipses [ETT06], or are given explicitly and are of arbitrary degree [EKW07, BEKS13]. The more general problems of computing the topology of arrangements of curves [BEKS11], and the topology of surfaces [BKS08, Ber14] are also being addressed under the EGC paradigm. These algorithms have been implemented under the `EXACUS` project [BEH$^+$05]. **(v)** The development of general geometric libraries such as `CGAL` [HHK$^+$07] or LEDA [MN95] exposes a range of issues peculiar to EGC (cf. Chapter 68). For instance, in EGC we want a framework where various number kernels and filters can be used for a single algorithm.

45.5 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 covertical 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, Sei98]. Sedgewick [Sed02] calls degeneracies the "bugbear of geometric algorithms." Degeneracy is a major cause of nonrobustness for two reasons. First, it presents severe difficulties for approximate arithmetic. Second, even under the EGC paradigm, implementers are faced with a large number of special degenerate cases that must be treated (this number grows exponentially in the dimension of the underlying space). For instance, the Voronoi diagram of the arrangements of three lines in $\mathbb{R}^3$ is characterized by five cases [EGL$^+$09]. In general, the types of Voronoi cells grow exponentially in the dimension [YSL12]. Thus there is a 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 S 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 versus General algorithm: A generic algorithm is one that is only guaranteed to be correct on generic inputs. A general algorithm is one that works correctly for all (legal) inputs. Beware that "general" and "generic" are used synonymously in the 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. There are two kinds of perturbations: symbolic (a.k.a. infinitesimal) or numeric. Informally, perturbation has the connotation of being random. But there are applications where we want to "control" these perturbations to achieve some properties: e.g., for a convex polytope A, we may like this to be an "outward perturbation" so that A' contains A. However, the term *controlled perturbation* has now taken a rather specific meaning in the framework introduced by Halperin et al. [HS98, Raa99]. The goal of controlled perturbation is to guarantee that a random δ-perturbation achieves a *sufficiently nondegenerate state* with high probability. Sufficient nondegeneracy here means that the underlying predicates can be computed with (say) IEEE floating point arithmetic, with a certification of nondegeneracy.

3. There is a *postprocessing issue*: although A' is "correct" in some technical sense, it may not necessarily produce the same outputs as an ideal algorithm A^*. This issue arises in symbolic rather than numeric perturbation. 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

There are three main methods for converting a generic algorithm to a general one:

Symbolic perturbation schemes (Blackbox sign evaluation) 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] and 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. This method of Simulation of Simplicity (SoS) is a special case of this scheme.

Numerical and controlled perturbation. In contrast to symbolic perturbation, we can make a random numerical perturbation to the input. Intuitively, we expect this results in nondegeneracy. But controlled perturbation [HS98, Raa99] in the sense of Halperin assumes other ingredients: one idea is that the perturbation must be "sufficiently nondegenerate" so that it could be "efficiently certified." To formalize this, consider this problem: *given any $\varepsilon > 0$ and input z, compute $\delta = \delta(\varepsilon) > 0$ such that a random δ-perturbation of z has probability $> 1/2$ of being **ε-nondegenerate**.* The set of δ-perturbations of z is $U_\delta(z) := \{z' : \|z' - z\| \leq \delta\}$, where $\|\cdot\|$ is any norm. Usually the infinity-norm is chosen (so $U_\delta(z)$ is a box). Assume that z is degenerate if $f(z) = 0$, where f is a given real function. Note that it is not enough to be "simply nondegenerate" (the probability of this is 1, trivially). We define ε-nondegenerate to mean $|f(u)| > \varepsilon$. How to choose this ε? It is dictated by a pragmatic assumption: for efficiency, assume we want to use the IEEE floating point arithmetic. Let $\widetilde{f}(z)$ denote the result of computing $f(z)$ in IEEE floating point. There are techniques (see below and [MOS11, appendix]) to compute a bound $B(f, M)$ such that for all $\|z\| \leq M$, if $|\widetilde{f}(z)| > B(f, M)$ then $|f(z)| > 0$, i.e., z is nondegenerate. Moreover, $B(f, M)$ can be computed in floating point, taking no more than big-Oh of the time to evaluate $f(z)$. Therefore, if we choose ε to be $B(f, M)$, we can verify that z is nondegenerate by checking that $|\widetilde{f}(z)| > B(f, M)$. If this check succeeds, we say z is ***certifiably nondegenerate*** and stop. If it fails, we repeat the process with another random δ-perturbation z'; but we are assured that the expected number of such repeats is at most one. In actual algorithms, we do not seek a monolithic perturbation of z, but view $z = (z_1, z_2, \ldots, z_n)$ to be a sequence of points where each z_i is an m-vector. We perturb the z_i's sequentially (in the order $i = 1, \ldots, n$). Inductively, suppose $(z'_1, \ldots, z'_{i-1})$ is certifiably nondegenerate. The perturbed z'_i must be chosen to extend this inductive hypothesis.

Perturbation toward a nondegenerate instance. A third approach is provided by Seidel [Sei98], 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 small 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 $(t, t^2, \ldots, t^n)$. This method can be regarded as symbolic or numeric, depending on whether ϵ is an infinitesimal or an actual numerical value.

We now elaborate on these methods. 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. An early work in this area is the Simulation of Simplicity (SoS) 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 1950s. 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 shows that a random choice of the b_i's is likely to yield nondegeneracy. Emiris, Canny, and Seidel [ECS97, Sei98] give a general result on the validity of linear perturbations, and apply it to common test polynomials.

Controlled perturbation was developed explicitly to be practical and to exploit fast floating point arithmetic. Our above assumption of the specific IEEE standard is only for simplicity: in general, we will need to consider floating point systems with L-bit mantissa as discussed in [MOS11]. By choosing L arbitrarily large, we can make the error bound $\varepsilon = B(f, M)$ arbitrarily close to 0. The idea of computing $B(f, M)$ goes back to Fortune and van Wyk [FW93a, FW96]. Given an expression E (typically a polynomial perhaps also with square-root), certain parameters such as m_E, ind_E, $\deg_E$, etc., can be recursively defined from the structure of E (e.g., [MOS11, Table 1]). They can even be computed in the chosen floating point system, costing no more than evaluating E in floating point. If $\widetilde{E}$ is the floating point value of E then [MOS11, Theorem 16] says that the error $|\widetilde{E} - E|$ is at most $(\mathrm{ind}_E + 1) \cdot \mathbf{u} \cdot m_E$, where $\mathbf{u} = 2^{-L-1}$ is the unit round-off error. If E is an expression for evaluating $f(z)$, then the bound $B(f, M)$ may be defined as $(\mathrm{int}_E + 1) \cdot \mathbf{u} \cdot m_E$.

We note two additional issues in controlled perturbation: (1) The above perturbation analysis operates at a predicate level. An algorithm execution will call these predicates many times on different inputs. We must translate the bounds and probabilities from predicate level to algorithm level. (2) The analysis assumes the input and perturbation are real numbers. Since the actual perturbations must be floating point numbers, we must convert our probability estimates based on real numbers into corresponding estimates for a discrete set of floating point numbers. Say u is a δ-perturbation of z. Let the $w = f\ell(u)$ be the floating point number closest to u (breaking ties arbitrarily). In our algorithm, we must be able to generate w with the probability p_w of the set $\{u \in U : f\ell(u) = w\}$. This issue was first addressed explicitly in Mehlhorn et al. [MOS11].

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 [ECS97]. 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.

Controlled perturbation was initially applied by Halperin et al. to arrangements for spheres, polyhedra and circles [HS98, Raa99, HL04]. This was extended to controlled perturbations for Delaunay triangulations [FKMS05], convex hulls in all dimensions [Kle04], and Voronoi diagrams of line segments [Car07]. Finally, Mehlhorn et al. [MOS11] provided the general analysis for all degeneracy predicates defined by multivariate polynomials, subsuming the previously studied cases.

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 57.1). 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. 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 *avoiding* degeneracies, there are some papers that propose to directly handle degeneracies. Burnikel, Mehlhorn, and Schirra [BMS95] describe the implementation of a line segment intersection algorithm and semidynamic 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 users when they have the option of writing a generic algorithm, especially when the general algorithm is very complex or not readily available. Unfortunately, property (iii) is the exception rather than the rule, especially in nonlinear and nonplanar settings. In illustration, consider the mildly nonlinear problem of computing the Voronoi diagram of convex polyhedra in $\mathbb{R}^3$. An explicit exact algorithm for this "Voronoi quest" is still elusive at the moment of this writing. Solution of a special case (Voronoi diagram of arbitrary lines in $\mathbb{R}^3$) was called a major milestone in Hemmer et al. [HSH10]. The fundamental barrier here is traced by [YSL12] to the lack of a complete degeneracy analysis, requiring 10 cases (with subcases) and nontrivial facts of algebraic geometry (the requisite algebraic geometry lemma for lines in $\mathbb{R}^3$ was provided by Everett et al. [ELLD09]). In short, users must weigh these opposing considerations for their particular problem (cf. [Sch94]).

45.6 SOFT EXACT COMPUTATION

EGC has emerged as a successful general solution to numerical nonrobustness for basic geometric problems, mainly planar and linear problems. In principle, this solution extends to nonlinear algebraic problems. But its complexity is a barrier in practice. For transcendental problems, the specter of noncomputable Zero problems (Section 45.4) must be faced. These considerations alone press us to weaken the notion of exactness. On top of this, the input is inherently inexact in the vast majority of geometric applications (e.g., in robotics or biology). All common physical constants, save the speed of light which is exact by definition, is known to at most 8 digits of accuracy. In such settings, exact computation could be regarded as a device to ensure robustness (treating the inexact input as nominally exact, Section 45.4). But this device is no longer reasonable when addressing regimes where exact computation is expensive or impossible. Numerical approaches [Yap09] promise to open up vast new domains in Computational Science and Engineering (CS&E) that are inaccessible to current techniques of Computational Geometry. This section describes current attempts to exploit numerics and achieve some weak notion (or soft notion) of exactness in regimes challenging for the current exact approach.

GLOSSARY

Traditional Numerical Algorithm: An algorithm described in the Real RAM model and then implemented in the ***Standard Model*** of Numerical Analysis [TB97]: namely, each primitive numerical operation returns a value with relative error at most $\mathbf{u}$, where $\mathbf{u} > 0$ is called the ***unit round-off error***. Of course, these errors will accumulate in subsequent steps.

Subdivision Model of Computation: A popular model for practitioners and can take many forms. Here we describe a fully adaptive version [LY11, Yap15] that supports rigorous soft algorithms using interval methods. Computational problems in a metric space X can often be reduced to "local computation" in small neighborhoods of X. Concretely, let $X = \mathbb{R}^d$, and $\square X$ denote the set of d-boxes in X, representing neighborhoods. A ***subdivision*** of any set $S \subseteq X$ is a finite collection C of sets with pairwise-disjoint interiors, whose union $\bigcup C$ equals S. Assume an operator that replaces any box B by a finite subdivision of B called split(B); each subbox in split(B) is a ***child*** of B. In this way, we form a ***subdivision tree*** $T(B)$ rooted at B where the parent-child relationship is determined by split operations. This is a dynamic tree that grows by splitting at leaves. The leaves of $T(B)$ form a subdivision of B. For example, for $d = 2$, split(B) is the set of four congruent subboxes sharing the midpoint of B, and $T(B)$ is called a ***quadtree***. To solve a problem in $B \subseteq X$, we grow the subdivision tree rooted at B by splitting at any leaf. Termination is determined by some numerical predicate on boxes. These predicates need only be evaluated to a precision determined by the depth, thus side-stepping exact computation. In general, we can replace boxes by other well-structured sets such as simplices; see [Yap15] for a treatment that includes subdivision in non-Euclidean spaces.

RESOLUTION EXACTNESS

Our return to numerics needs new approaches: after all, we were driven to the exact approach in the first place because the ***Traditional Numerical Algorithms*** led to insoluble nonrobustness issues. Such algorithms do not have an a priori guarantee of correctness, but depend on an a posteriori error analysis. What we seek are algorithms with a priori correctness guarantees, and which can dynamically adjust their arbitrary precision and steps depending on the input instance. Informally, we call this mode of numerical computation "Soft Exact Computation."

The approximate version of an arbitrary problem is obtained by introducing a new ***resolution parameter*** $\varepsilon > 0$, in addition to its normal parameters. We must not identify ε with the unit round-off error $\mathbf{u}$ of the ***Standard Model*** of numerical analysis; intuitively, we want ε to bound the output error. In approximation algorithms, this has a standard interpretation called ε-***approximation*** for optimization problems. It says that the output value should be within a relative error of ε of the optimal. For example, for shortest path problems, the output path length is $\leq (1+\varepsilon)$ times the shortest length. If no path exists, the optimal length is ∞. Observe that such an ε-approximation algorithm implicitly solves the same path-existence predicate as the original algorithm: it outputs ∞ iff there is no path. Since this predicate requires exact computation, it is not really what we want. In fact, we want to approximate even the nonoptimization problems; in particular, what are "ε-approximate predicates"? In general, this is not a meaningful thought. However we can define a sensible notion in case of ***numerical predicates***: if $f : X \to \mathbb{R}$ is a continuous function over some metric space X, then it defines a numerical predicate $C_f : X \to \{\mathtt{true}, \mathtt{false}\}$ given by $C_f(x) = \mathtt{true}$ iff $f(x) > 0$.

To be concrete, consider the robot motion planning problem (Chapter 50) for some fixed robot in $\mathbb{R}^3$: given $x = (A, B, \Omega)$, we must find a path from start configuration A to goal configuration B while avoiding some polygonal set $\Omega \subseteq \mathbb{R}^3$. Let X be the set of all such inputs x. There is[2] a nontrivial path-existence predicate here: does input $x = (A, B, \Omega)$ admit a path? Let $f(x)$ be defined as the minimum clearance of an Ω-avoiding path from A to B if one exists; $f(x) = 0$ if there is no such path. Thus C_f is the path-existence predicate. Following [WCY15, LCLY15, Yap15], we call a function of the form $\widetilde{C}_f : X \times \mathbb{R}_{>0} \to \{\mathtt{true}, \mathtt{false}\}$ ***resolution-exact*** (or ε-exact) version of C_f if there exists a $K > 1$ such that for all $(x, \varepsilon) \in X \times \mathbb{R}_{>0}$, we have the following:

(T) If $f(x) > K\varepsilon$, then $\widetilde{C}_f(x, \varepsilon) = \mathtt{true}$.

(F) If $f(x) < \varepsilon/K$, then $\widetilde{C}_f(x, \varepsilon) = \mathtt{false}$.

The conditions (T) and (F) are nonexhaustive: in case $f(x) \in [\varepsilon/K, K\varepsilon]$, $\widetilde{C}_f(x)$ can output either answer. Because of this, we say that the ε-exactness problem is

[2] Current robotics literature ignores the path-existence predicate and formulates their correctness criteria for path planning, but *only* for inputs that admit a path; two such criteria are "resolution completeness" and "probabilistic completeness" [LaV06].

indeterminate. Indeterminacy allows us to bypass the specter of the Zero problem because the computability of $\widetilde{C}_f$ is not in question as long as we can arbitrarily approximate f.

Another crucial step in transitioning to numerical approximation is to abandon the Real RAM model implicit in Traditional Numerical Algorithms. The alternative is conveniently captured by the ***Subdivision Model***. We need techniques from interval arithmetic [Moo66]: assume $X = \mathbb{R}^d$ and $\Box\mathbb{R}^d$ is the set of full-bodied axes-parallel boxes. If $f : X \to \mathbb{R}$ and $\Box f : \Box X \to \Box\mathbb{R}$, we call $\Box f$ a ***soft version*** of f if $\Box f$ is ***conservative*** (i.e., $f(B) \subseteq \Box f(B)$ for $B \in \Box\mathbb{R}^d$) and ***convergent*** (i.e., $\lim_{i\to\infty} \Box f(B_i) = f(p)$ for any sequence $\{B_i : i \geq 0\}$ of boxes that monotonically converges to a point $p \in X$). Such soft functions are relatively easy to design and implement for motion planning problems, leading to efficient and practical ε-exact planners [WCY15]. Soft exact algorithms have been developed for Voronoi diagrams [BPY16], meshing curves and surfaces [PV04, LY11, LYY12], and arrangement of curves [LSVY14].

The approach can be systematically extended to the first-order theory of reals as in Gao et al. [GAC12b]: given a Tarski formula φ, there are standard ways to bring it into a positive prenex form whose matrix is a Boolean combination of atomic predicates of the form $t(\mathbf{x}) > 0$ or $t(\mathbf{x}) \geq 0$ where $t(\mathbf{x})$ are terms (i.e., polynomials). Positive means that there are no negations in φ. We can enrich Tarski's language by allowing terms that include functions from some subset F of real functions that are computable in the sense of computable analysis [Wei00]. Call these F***-formulas***. For any $\delta \geq 0$, the δ***-strengthened form*** of φ, denoted by $\varphi^{+\delta}$, is obtained by replacing the atomic predicate $t(\mathbf{x}) > 0$ by $t(\mathbf{x}) > \delta$ and $t(\mathbf{x}) \geq 0$ by $t(\mathbf{x}) \geq \delta$ in the positive form. Clearly, $\varphi^{+\delta}$ implies φ. Next, an F-formula φ is said to be ***bounded*** if each occurrence of a quantifier Q_i $(i = 1, 2, \ldots)$ appears in the form $(Q_i x_i \in [u_i, v_i])[\ldots]$ where $u_i \leq v_i$ are both terms involving variables x_j, for $j < i$. The δ***-decision problem*** is this: *given a bounded F-sentence φ and $\delta > 0$, decide whether φ is true or $\varphi^{+\delta}$ is false.* Their main result in [GAC12b] is that this problem is decidable. Prima facie, this is surprising since the decision problem for F-sentences is already undecidable if the $\sin(\cdot)$ function is included in F [Ric68]. It is less surprising when we realize that δ-decision is indeterminate, not the usual "hard" decision involving the Zero problem for F-terms. The δ-strengthening is one of several ways to strengthen (or weaken) numerical predicates; the above notion of ε-exactness is a variant better suited to nonnegative functions like clearance $f(x)$ and to the physical interpretations of robot uncertainty. It is also possible to strengthen using relative errors. Gao et al. [GAC12a] address the more practical case of δ-decision for bounded existential F-sentences. They further introduce a subdivision algorithm with interval evaluations (in the terminology of verification literature, this is called the ***DPLL-Interval Constraint Propagation*** (or DPLL/ICP)). It is proved that the DPLL/ICP algorithm can δ-decide bounded existential F-sentences within the complexity class NP^P assuming that the functions in F are P-computable.

45.7 OPEN PROBLEMS

1. The main theoretical question in EGC is whether the Constant Zero Problem for Ω_4 is decidable. This is decidable if Schanuel's conjecture is true. Baker's

theory gives a positive answer to a very special subproblem. This is expected to be very deep, so deciding any nontrivial subproblem would be of interest. A 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 Revised Uniformity Conjecture of Richardson and Sonbaty [RES06] is a useful starting point.

3. Give an 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 nonuniform 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), but rather allow features to collapse (triangles can degenerate to a vertex), but disallow inversion (triangles cannot flip its orientation).

5. Extend the control perturbation technique to more general classes of expressions, including rational functions, square-root functions, analytic functions [MOS11].

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

7. Design soft exact algorithms for kinodynamic planning and for nonholonomic planning in robotics. Known theoretical algorithms are far from practical (and practical ones are nonrigorous). Even when subdivision is used, such algorithms use exact ("hard") predicates.

8. Develop techniques for nontrivial complexity analysis of subdivision algorithms in computation geometry. Since such algorithms are adaptive, we would like to use complexity parameters based on the geometry of the input instance such as feature size or separation bounds.

45.8 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 [Sch00], Yap and Dubé [YD95] and Fortune [For93]. Robust geometric modelers are surveyed in [PCH+95].

RELATED CHAPTERS

Chapter 26: Convex hull computations
Chapter 28: Arrangements
Chapter 31: Shortest paths and networks
Chapter 37: Computational and quantitative real algebraic geometry
Chapter 41: Ray shooting and lines in space
Chapter 50: Algorithmic motion planning
Chapter 51: Robotics
Chapter 57: Solid modeling
Chapter 67: Software
Chapter 68: Two computational geometry libraries: LEDA and CGAL

REFERENCES

[AKY04] T. Asano, D. Kirkpatrick, and C. Yap. Pseudo approximation algorithms, with applications to optimal motion planning. *Discrete Comput. Geom.*, 31:139–171, 2004.

[Bak75] A. Baker. *Transcendental Number Theory.* Cambridge University Press, 1975.

[BBP01] H. Brönnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Appl. Math.*, 109:25–47, 2001.

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

[BCSM$^+$06] J.-D. Boissonnat, D. Cohen-Steiner, B. Mourrain, G. Rote, and G. Vegter. Meshing of surfaces. Chap. 5 in J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, Springer, Berlin, 2006.

[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. 10th European Sympos. Algorithms*, vol. 2461 of *LNCS*, pages 174–186, Springer, Berlin, 2002.

[BEH$^+$05] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, L. Kettner, K. Mehlhorn, J. Reichel, S. Schmitt, E. Schömer, and N. Wolpert. EXACUS: Efficient and Exact Algorithms for Curves and Surfaces. In *Proc. 13th European Sympos. Algorithms*, vol. 3669 of *LNCS*, pages 155–166, Springer, Berlin, 2005.

[BEKS11] E. Berberich, P. Emeliyanenko, A. Kobel, and M. Sagraloff. Arrangement computation for planar algebraic curves. In *Proc. Int. Workshop Symbolic-Numeric Comput.*, pages 88–98, ACM Press, 2011.

[BEKS13] E. Berberich, P. Emeliyanenko, A. Kobel, and M. Sagraloff. Exact symbolic-numeric computation of planar algebraic curves. *Theor. Comput. Sci.*, 491:1–32, 2013.

[Ber14] E. Berberich. Robustly and efficiently computing algebraic curves and surfaces. In *Proc. 4th Int. Congr. Math. Software*, vol. 8592 of *LNCS*, pages 253–260, Springer, Berlin, 2014.

[BFM$^+$09] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55:14–28, 2009.

[BFMS99] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Exact efficient geometric computation made easy. In *Proc. 15th ACM Sympos. Comput. Geom.*, pages 341–450, ACM Press, 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 Sympos. Comput. Geom.*, pages 175–183, ACM Press, 1998.

[BJMM93] M.O. Benouamer, P. Jaillon, D. Michelucci, and J.-M. Moreau. A lazy arithmetic library. In *Proc. 11th IEEE Sympos. Comp. Arithmetic*, pages 242–269, 1993.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42:269–291, 1995.

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

[BKS08] E. Berberich, M. Kerber, and M. Sagraloff. Exact geometric-topological analysis of algebraic surfaces. In *Proc. 24th Sympos. Comput. Geom.*, pages 164–173, ACM Press, 2008.

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

[BMS94] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In *Proc. 2nd Eur. Sympos. Algo.*, vol. 855 of *LNCS*, pages 227–239, Springer, Berlin, 1994.

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

[BPY16] H. Bennett, E. Papadopoulou, and C. Yap. Planar minimization diagrams via subdivision with applications to anisotropic Voronoi diagrams. In *Proc. Eurographics Sympos. Geom. Process.*, 2016.

[BS00] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. *Comput. Geom.*, 16:35–52, 2000.

[BSS$^+$16] R. Becker, M. Sagraloff, V. Sharma, J. Xu, and C. Yap. Complexity analysis of root clustering for a complex polynomial. In *Proc. Int. Sympos. Symb. Algebr. Comput.*, pages 71–78, ACM Press, 2016.

[BSSY15] R. Becker, M. Sagraloff, V. Sharma, and C. Yap. A simple near-optimal subdivision algorithm for complex root isolation based on the pellet test and Newton iteration. Preprint, arXiv:1509.06231, 2015.

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

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

[BY09] W.D. Brownawell and Chee K. Yap. Lower bounds for zero-dimensional projections. In *Proc. 34th Int. Sympos. Symb. Algebr. Comput.*, pages 79–86, ACM Press, 2009.

[Can88] J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertion Award Series, MIT Press, Cambridge, 1988.

[Car07] M. Caroli. Evaluation of a generic method for analyzing controlled-perturbation algorithms. Master's thesis, University of Saarland, 2007.

[CCK$^+$06] E.-C. Chang, S.W. Choi, D. Kwon, H. Park, and C.K. Yap. Shortest paths for disc obstacles is computable. *Internat. J. Comput. Geom. Appl.*, 16:567–590, 2006.

[CDR92] J.F. Canny, B. Donald, and E.K. Ressler. A rational rotation method for robust geometric algorithms. *Proc. 8th Sympos. Comp. Geom.*, pages 251–160, ACM Press, 1992.

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

[Cla92] K.L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd IEEE Sympos. Found. Comp. Sci.*, pages 387–395, 1992.

[CM93] J.D. Chang and V.J. Milenkovic. An experiment using LN for exact geometric computations. *Proc. 5th Canad. Conf. Comput. Geom.*, pages 67–72, 1993.

[CSY97] J. Choi, J. Sellen, and C. Yap. Approximate Euclidean shortest paths in 3-space. *Internat. J. Comput. Geom. Appl.*, 7:271–295, 1997.

[CZ99] J. Comes and M. Ziegelmann. An easy to use implementation of linear perturbations within CGAL. In *Proc. 3rd Workshop Algorithms Eng.*, vol. 1668 of *LNCS*, pages 169–182, Springer, Berlin, 1999.

[DEMY02] Z. Du, M. Eleftheriou, J. Moreira, and C. Yap. Hypergeometric functions in exact geometric computation. *Electron. Notes Theor. Comput. Sci.*, 66:53–64, 2002.

[DP99] O. Devillers and F.P. Preparata. Further results on arithmetic filters for geometric predicates. *Comput. Geom.*, 13:141–148, 1999.

[DS88] D.P. Dobkin and D. Silver. Recipes for Geometry & Numerical Analysis – Part I: An empirical study. In *Proc. 4th Sympos. Comput. Geom.*, pages 93–105, ACM Press, 1988.

[DY93] T. Dubé and C.K. Yap. A basis for implementing exact geometric algorithms (extended abstract). Unpublished manuscript, 1993.
`http://cs.nyu.edu/exact/doc/basis.ps.gz`.

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

[ECS97] I.Z. Emiris, J.F. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19:219–242, 1997.

[EGL$^+$09] H. Everett, C. Gillot, D. Lazard, S. Lazard, and M. Pouget. The Voronoi diagram of three arbitrary lines in $\mathbb{R}^3$. In *Abstracts of 25th Eur. Workshop Comput. Geom.*, Bruxelles, 2009.

[EKSW06] A. Eigenwillig, L. Kettner, E. Schömer, and N. Wolpert. Complete, exact, and efficient computations with cubic curves. *Comput. Geom.*, 35:36–73, 2006.

[EKW07] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and exact geometric analysis of real algebraic plane curves. In *Int. Sympos. Symb. Alge. Comp.*, pages 151–158, ACM Press, 2007.

[ELLD09] H. Everett, D. Lazard, S. Lazard, and M.S. El Din. The Voronoi diagram of three lines. *Discrete Comput. Geom.*, 42:94–130, 2009.

[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.

[EMT10] I.Z. Emiris, B. Mourrain, and E.P. Tsigaridas. The DMM bound: Multivariate (aggregate) separation bounds. In *35th Int. Sympos. Symb. Algebr. Comput.*, pages 243–250, ACM Press, 2010.

[ETT06] I.Z. Emiris, E.P. Tsigaridas, and G.M. Tzoumas. The predicates for the Voronoi diagram of ellipses. In *Proc. 22nd Sympos. Comput. Geom.*, pages 227–236, ACM Press, 2006.

[FHL$^+$07] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Software*, 33: article 13, 2007.

[FKMS05] S. Funke, C. Klein, K. Mehlhorn, and S. Schmitt. Controlled perturbation for Delaunay triangulations. In *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, pages 1047–1056, 2005.

[FM91] S.J. Fortune and V.J. Milenkovic. Numerical stability of algorithms for line arrangements. In *Proc. 7th Sympos. Comput. Geom.*, pages 334–341, ACM Press, 1991.

[FMN05] S. Funke, K. Mehlhorn, and S. Näher. Structural filtering: A paradigm for efficient and exact geometric programs. *Comput. Geom.*, 31:179–194, 2005.

[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, New York, 1987.

[For89] S.J. Fortune. Stable maintenance of point-set triangulations in two dimensions. In *Proc. 30th IEEE Sympos. Found. Comp. Sci.*, pages 494–499, 1989.

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

[For93] S.J. Fortune. Progress in computational geometrey. In R. Martin, editor, *Directions in Geometric Computing*, pages 81–127, Information Geometers Ltd., Winchester, 1993.

[For95] S.J. Fortune. Numerical stability of algorithms for 2-d Delaunay triangulations. *Internat. J. Comput. Geom. Appl.*, 5:193–213, 1995.

[For97] S.J. Fortune. Polyhedral modeling with multiprecision integer arithmetic. *Comput.-Aided Des.*, 29:123–133, 1997.

[For99] S.J. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. *Discrete Comput. Geom.* 22:593–618, 1999.

[Fun97] S. Funke. *Exact Arithmetic Using Cascaded Computation*. Master's thesis, Max Planck Institute for Computer Science, Saarbrücken, 1997.

[FW93a] S.J. Fortune and C.J. van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Sympos. Comput. Geom.*, pages 163–172, ACM Press, 1993.

[FW93b] S.J. Fortune and C.J. van Wyk. *LN User Manual*. AT&T Bell Laboratories, 1993.

[FW96] S.J. Fortune and C.J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15:223–248, 1996.

[GAC12a] S. Gao, J. Avigad, and E.M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In *Porc. 6th Int. Joint Conf. Automated Reasoning*, vol. 7364 of *LNCS*, pages 286–300, Springer, Berlin 2012.

[GAC12b] S. Gao, J. Avigad, and E.M. Clarke. Delta-decidability over the reals. In *Proc. 27th IEEE Sympos. Logic in Comp. Sci.*, pages 305–314, 2012.

[Gae99] B. Gärtner. Exact arithmetic at low cost—a case study in linear programming. *Comput. Geom.*, 13:121–139, 1999.

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

[GHS01] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th Sympos. Comput. Geom.*, pages 264–273, ACM Press, 2001.

[GM95] L.J. Guibas and D. Marimont. Rounding arrangements dynamically. In *Proc. 11th Sympos. Comput. Geom.*, pages 190–199, ACM Press, 1995.

[GS00] B. Gärtner and S. Schönherr. An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proc. 16th Sympos. Comput. Geom.*, pages 110–118, ACM Press, 2000.

[GSS89] L.J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proc. 5th Sympos. Comput. Geom.*, pages 208–217, ACM Press, 1989.

[GY86] D.H. Greene and F.F. Yao. Finite-resolution computational geometry. In *Proc. 27th IEEE Sympos. Found. Comp. Sci.*, pages 143–152, 1986.

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

[Hel01b] M. Held. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Comput. Geom.*, 18:95–123, 2001.

[HHK88] C.M. Hoffmann, J.E. Hopcroft, and M.S. Karasick. Towards implementing robust geometric computations. In *Proc. 4th Sympos. Comput. Geom.*, pages 106–117, ACM Press, 1988.

[HHK$^+$07] S. Hert, M. Hoffmann, L. Kettner, S. Pion, and M. Seel. An adaptable and extensible geometry Kernel. *Comput. Geom.*, 38:16–36, 2007.

[HL04] D. Halperin and E. Leiserowitz. Controlled perturbation for arrangements of circles. *Internat. J. Comput. Geom. Appl.*, 14:277–310, 2004.

[Hob99] J.D. Hobby. Practical segment intersection with finite precision output. *Comput. Geom.*, 13:199–214, 1999.

[HS98] D. Halperin and C.R. Shelton. A perturbation scheme for spherical arrangements with applications to molecular modeling. *Comput. Geom.*, 10:273–288, 1998.

[HSH10] M. Hemmer, O. Setter, and D. Halperin. Constructing the exact Voronoi diagram of arbitrary lines in three-dimensional space. In *Proc. 18th Eur. Sympos. Algorithms* vol. 6346 of *LNCS*, pages 398–409, Springer, Berlin, 2010.

[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. *Comput. Geom.*, 4:283–292, 1994.

[KFC$^+$01] S. Krishnan, M. Foskey, T. Culver, J. Keyser, and D. Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. In *17th Sympos. Comput. Geom.*, pages 274–283, ACM Press, 2001.

[Kle04] C. Klein. *Controlled Perturbation for Voronoi Diagrams*. Master's thesis, University of Saarland, 2004.

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

[KLPY99] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A Core library for robust numerical and geometric computation. In *15th Sympos. Comput. Geom.*, p. 351–359, ACM Press, 1999.

[KMP$^+$07] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computation. *Comput. Geom.*, 40:61–78, 2007.

[KW98] L. Kettner and E. Welzl. One sided error predicates in geometric computing. In *Proc. 15th IFIP World Computer Congress*, pages 13–26, 1998.

[LaV06] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LCLY15] Z. Luo, Y.-J. Chiang, J.-M. Lien, and C. Yap. Resolution exact algorithms for link robots. In *Proc. 11th Workshop Algo. Found. Robot.*, vol. 107 of *Springer Tracts in Advanced Robotics*, pages 353–370, 2015.

[LPT98] G. Liotta, F.P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28:864–889, 1998.

[LPY04] C. Li, S. Pion, and C.K. Yap. Recent progress in Exact Geometric Computation. *J. Log. Algebr. Program.*, 64:85–111, 2004.

[LSVY14] J.-M. Lien, V. Sharma, G. Vegter, and C. Yap. Isotopic arrangement of simple curves: An exact numerical approach based on subdivision. In *Proc. 4th Int. Congr. Math. Software*, vol. 8592 of *LNCS*, pages 277–282, Springer, Berlin, 2014.

[LY01] C. Li and C.K. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 496–505, 2001.

[LY11] L. Lin and C. Yap. Adaptive isotopic approximation of nonsingular curves: The parameterizability and nonlocal isotopy approach. *Discrete Comput. Geom.*, 45:760–795, 2011.

[LYY12] L. Lin, C. Yap, and J. Yu. Non-local isotopic approximation of nonsingular surfaces. *Comput.-Aided Des.*, 45:451–462, 2012.

[Mic95] D. Michelucci. An epsilon-arithmetic for removing degeneracies. In *Proc. 12th IEEE Sympos. Comp. Arith.*, pages 230–237, 1995.

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

[Mil89] V. Milenkovic. Double precision geometry: A general technique for calculating line and segment intersections using rounded arithmetic. In *Proc. 30th IEEE Sympos. Found. Comp. Sci.*, pages 500–506, 1989.

[MM93] V.J. Milenkovic and Ve. Milenkovic. Rational orthogonal approximations to orthogonal matrices. *Proc. 5th Canad. Conf. on Comp. Geom.*, pages 485–490, Waterloo, 1993.

[MN90] V. Milenkovic and L.R. Nackman. Finding compact coordinate representations for polygons and polyhedra. In *Proc. 6th Sympos. Comp. Geom.*, pages 244–252, ACM Press, 1990.

[MN95] K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. *Comm. ACM*, 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 Sympos. Comput. Geom.*, pages 159–165, ACM Press, 1996.

[Moo66] R.E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, 1966.

[MOS11] K. Mehlhorn, R. Osbild, and M. Sagraloff. A general approach to the analysis of controlled perturbation algorithms. *Comput. Geom.*, 44:507–528, 2011.

[MS15] M. Mörig and S. Schirra. Precision-driven computation in the evaluation of expression-dags with common subexpressions: Problems and solutions. In *Proc. 6th Math. Aspects Comp. Info. Sci.*, vol. 9582 of *LNCS*, pages 451–465, Springer, Berlin, 2015.

[MW96] A. 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, Wellesley, 1996.

[Neu97] M.A. Neuhauser. Symbolic perturbation and special arithmetics for controlled handling of geometric degeneracies. In *Proc. 5th Int. Conf. Central Europe Comput. Graphics Visualization*, pages 386–395, 1997.

[OTU87] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *Proc. 3rd Sympos. Comput. Geom.*, pages 119–125, ACM Press, 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. NSF Design and Manufacturing Grantees Conference*, pages 139–140, 1995.

[PV04] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Proc. Eurographics Sympos. Geom. Process.*, pages 245–254, ACM Press, 2004.

[PY01] V.Y. Pan and Y. Yu. Certification of numerical computation of the sign of the determinant of a matrix. *Algorithmica*, 30:708–724, 2001.

[PY06] S. Pion and C. K. Yap. Constructive root bound method for k-ary rational input numbers. *Theoret. Comput. Sci.* 369:361–376, 2006.

[Raa99] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th Sympos. Comput. Geom.*, pages 163–172, ACM Press, 1999.

[RES06] D. Richardson and A. El-Sonbaty. Counterexamples to the uniformity conjecture. *Comput. Geom.*, 33:58–64, 2006.

[Ric68] D. Richardson. Some undecidable problems involving elementary functions of a real variable. *J. Symb. Log.*, 33:514–520, 1968.

[Ric97] D. Richardson. How to recognize zero. *J. Symbolic Comput.*, 24:627–645, 1997.

[Ric07] D. Richardson. Zero tests for constants in simple scientific computation. *Math. Comput. Sci.*, 1(:21–37, 2007.

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

[Sch00] S. Schirra. Robustness and precision issues in geometric computation. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632, Elsevier, Amsterdam, 2000.

[Sch00] E.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. Comput.*, 29:1577–1595, 2000.

[Sed02] R. Sedgewick. *Algorithms in C: Part 5: Graph Algorithms*, 3rd edition. Addison-Wesley, Boston, 2002.

[See01] M. Seel. *Planar Nef Polyhedra and Generic High-dimensional Geometry*. Ph.D. thesis, Universität des Saarlandes, 2001.

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

[Sei98] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete Comput. Geom.*, 19:1–17, 1998.

[She96] J.R. Shewchuk. Robust adaptive floating-point geometric predicates. In *Proc. 12th Sympos. Comput. Geom.*, pages 141–150, ACM Press, 1996.

[SI89a] K. Sugihara and M. Iri. A solid modeling system free from topological inconsistency. *J. Inf. Process.*, 12:380–393, 1989.

[SI89b] K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite precision arithmetic. *Appl. Math. Lett.*, 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*, 80:1471–1484, 1992.

[SIII00] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation—an approach to robust geometric algorithms. *Algorithmica*, 27:5–20, 2000.

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

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

[Sug89] K. Sugihara. On finite-precision representations of geometric objects. *J. Comput. Syst. Sci.*, 39:236–247, 1989.

[Sug11] K. Sugihara. Robust geometric computation based on the principle of independence. *Nonlinear Theory Appl., IEICE*, 2:32–42, 2011.

[TB97] L.N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[Ull90] C. Ullrich, editor. *Computer Arithmetic and Self-validating Numerical Methods*. Academic Press, Boston, 1990.

[WCY15] C. Wang, Y.J. Chiang, and C. Yap. On soft predicates in subdivision motion planning. *Comput. Geom.*, 48:589–605, 2015.

[Wei00] K. Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[Wei02] R. Wein. High level filtering for arrangements of conic arcs. In *Proc. 10th Eur. Sympos. Algorithms*, vol. 2461 of *LNCS*, pages 884–895, Springer, Berlin, 2002.

[Yap90a] C.K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.

[Yap90b] C.K. Yap. Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.*, 10:349–370, 1990.

[Yap97] C.K. Yap. Towards exact geometric computation. *Comput. Geom.*, 7:3–23, 1997.

[Yap98] C.K. Yap. A new number core for robust numerical and geometric libraries. In *Abstract 3rd CGC Workshop Comput. Geom.*, Brown University, Providence, 1998. `http://www.cs.brown.edu/cgc/cgc98/home.html`.

[Yap00] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford Univ. Press, 2000.

[Yap09] C.K. Yap. In praise of numerical computation. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, vol. 5760 of *LNCS*, pages 308–407. Springer, Berlin, 2009.

[Yap15] C.K. Yap. Soft subdivision search and motion planning, II: Axiomatics. In *Proc. 9th Conf. Frontiers in Algorithmics*, vol 9130 of *LNCS*, pages 7–22, Springer, Berlin, 2015.

[YD95] C.K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, 2nd edition, pages 452–492, World Scientific, Singapore, 1995.

[YSL12] C. Yap, V. Sharma, and J.M. Lien. Towards exact numerical Voronoi diagrams. In *Proc. 9th Int. Sympos. Voronoi Diagrams in Sci. Eng.*, pages 2–16, 2012.

[Yu92] J. Yu. *Exact Arithmetic Solid Modeling.* Ph.D. dissertation, Department of Computer Science, Purdue University, West Lafayette, 1992.

[YYD$^+$10] J. Yu, C. Yap, Z. Du, S. Pion, and H. Brönnimann. The design of Core 2: A library for exact numeric computation in geometry and algebra. In *Proc. 3rd Int. Congr. Math. Software*, vol. 6327 of *LNCS*, pages 121–141, Springer, Berlin, 2010.