# Efficient Perturbations for Handling Geometric Degeneracies

Ioannis Z. Emiris,[1] John F. Canny,[2] and Raimund Seidel[3]

March 21, 1996

**Abstract.** This article defines input perturbations so that an algorithm designed under certain restrictions on the input can execute on arbitrary instances. A syntactic definition of perturbations is proposed and certain properties are specified under which an algorithm executed on perturbed input produces an output from which the exact answer can be recovered. A general framework is adopted for linear perturbations, which are efficient from the point of view of worst-case complexity. The deterministic scheme of Emiris and Canny [1] was the first efficient scheme and is extended in a consistent manner, most notably to the InSphere primitive. We introduce a variant scheme, applicable to a restricted class of algorithms, which is almost optimal in terms of algebraic as well as bit complexity. Neither scheme requires any symbolic computation and both are simple to use as illustrated by our implementation of a convex hull algorithm in arbitrary dimension. Empirical results and a concrete application in robotics are presented.

**Key words.** Input degeneracy, efficient perturbations, algorithm implementation, general-dimensional convex hulls.

## 1 Introduction

Algorithms in computational geometry typically make certain assumptions about the input. The treatment of cases violating these assumptions is tedious and intricate, thus seldom included in the theoretical discussion, yet it remains a nontrivial matter for implementors. For instance, in constructing convex hulls in $d$ dimensions, certain algorithms suppose that no $d+1$ points lie on the same hyperplane. A sweep-line algorithm in the plane may even require that no two points are covertical. This article describes a general approach to eliminate the need of explicitly dealing with some of these special cases.

The first contribution of this article is a syntactic definition of perturbations as curves rooted at input instances. A limiting process is employed to define perturbed instances, thus conforming to the intuitive notion of infinitesimal change. We discuss how to recover the answer to the original problem from the output on perturbed input, either directly or after some case-specific postprocessing. Some general techniques for designing and evaluating efficient perturbations for a wide class of geometric primitives are suggested.

The main drawback of previous approaches [2, 3] is that they increase the worst case asymptotic complexity by an exponential factor in the space dimension, which makes them unattractive for algorithms in general dimension. The deterministic perturbation of Emiris and Canny [1] was the

[1] Projet SAFIR, I.N.R.I.A., B.P. 93, 06902 Sophia-Antipolis, France; emiris@sophia.inria.fr.

[2] Computer Science Division, University of California, Berkeley CA 94720, USA; jfc@cs.Berkeley.edu.

[1,2] Supported by a Packard Foundation Fellowship and by NSF PYI Grant IRI-8958577.

[3] Universität des Saarlandes, FB 14, Informatik, Geb. 36, Postfach 151150, 66041 Saarbrücken, Germany; seidel@cs.uni-sb.de. Supported by NSF PYI Grant CCR-9058440.

first efficient scheme in the sense that the algebraic and bit complexity overheads are respectively at most logarithmic and polynomial in the dimension. It was applied to the Orientation and Transversality primitives. The second contribution of this article is to extend the applicability of this scheme to the InSphere and Ordering primitives. Furthermore, we propose a new variant that applies to Orientation, Transversality as well as Ordering and reduces the bit complexity overhead to a logarithmic factor in the dimension.

In addition to their efficiency, our schemes require no symbolic computation. For rational inputs, almost all arithmetic can be carried out over a finite field and all intermediate quantities grow in a quasi-linear fashion with the dimension. We discuss modular arithmetic which is a very efficient way to carry out exact integer computation. We also explain how the requirement on exact computation does not exclude floating-point inputs. Both schemes are simple to implement. To illustrate this claim, our third contribution is an implementation of the Beneath-Beyond algorithm that uses the second scheme to construct the facet structure of convex hulls in arbitrary dimension and to compute their volume. The issue of postprocessing is closely examined in this context and experimental results are reported. Finally, we sketch an application to computing the pose of industrial parts falling on a conveyor belt.

This article includes, in final form, certain results presented in [4]; it is organized as follows. The next section defines the computational model, the problem at hand, the notion of perturbations and how they are implemented and examines some positive and negative consequences of applying them. Section 3 is a comparative study of previous work on handling degeneracies. Linear perturbations are discussed in Section 4 where sufficient conditions for establishing the validity of particular schemes are explored. Section 5 discusses general methods for evaluating primitives on perturbed input as well as more efficient techniques for specific classes of primitives. The two perturbation schemes of interest are shown to be valid with respect to four common primitives in Section 6 and the complexity claims are demonstrated. Section 7 presents our implementation of the Beneath-Beyond algorithm and a specific application to robotics. The conclusion summarizes the main results and suggests some open questions.

# 2 Definitions

Our approach is largely based on those of Edelsbrunner [2] and Yap [3]. The formalization of Emiris and Canny [1] focuses on the desired *effects* of the perturbation. The current setting, first introduced by Seidel [5], offers a *syntactic* definition of perturbations. All approaches, however, are essentially equivalent and lead, for a given set of primitives, to the same requirements on the perturbation schemes.

## 2.1 Computational model

Our model is the real Random Access Machine (RAM) of [6]. The *input* is organized as a set of $n$ vectors in $\mathbb{R}^d$, where $n \geq d > 0$ and the $i$-th vector is $x_i = (x_{i,1}, \ldots, x_{i,d})$ for $1 \leq i \leq n$, $1 \leq j \leq d$. The four basic operations $\{+, -, \times, /\}$ are assumed to be exact between real numbers, where the operands are constants, input quantities or have been computed previously. Branching occurs at tests against zero of an input or computed quantity and is three-way, depending on the sign of the tested value.

The set of arithmetic operations computing a branch expression together with the corresponding test is referred to as a *primitive*. A typical primitive used in sorting, called Ordering, is the comparison of coordinates. The branch polynomial is $f = x_{ij} - x_{kj}$ for $1 \leq i \neq k \leq n$. Another

is the Orientation primitive. For the planar convex hull problem the branch polynomial is the determinant of

$$\Lambda_3 = \begin{bmatrix} 1 & x_{i_1,1} & x_{i_1,2} \\ 1 & x_{i_2,1} & x_{i_2,2} \\ 1 & x_{i_3,1} & x_{i_3,2} \end{bmatrix}$$

where $x_{i_1}, x_{i_2}, x_{i_3}$ are distinct input points. This test decides on which side of the line $(x_{i_1}, x_{i_2})$ the third point $x_{i_3}$ lies. More primitives, including InSphere, are discussed below. The real RAM produces a unique *output* for any given input instance. We refer to a *program*, which is a sequence of instructions that implements a specific algorithm, and to an *execution path* in the program or algorithm, which is the sequence of instructions executed on a particular input instance.

We make use of two complexity measures. Under the *algebraic model* the total cost of a program equals the number of instructions in the longest execution path. More realistically, we must keep track of the operands' bit size: under the *bit model* only the input, output and branching instructions are assigned unit cost. For integers of size $\mathcal{O}(b)$, addition and subtraction require $\mathcal{O}(b)$ bit operations while multiplication and division require $\mathcal{O}(b \log b \log\log b)$ bit operations [7]. We shall use $M(b) = \mathcal{O}(b \log b \log\log b)$ as an upper bound on the bit complexity of any arithmetic operation between any two integers of size $\mathcal{O}(b)$. The total bit complexity of a real RAM program equals the sum of the costs of every instruction on an execution path, maximized over all paths.

It must be underlined that exact computation is indispensable in any implementation of our methods, since resolving the special, or degenerate, cases relies on having no roundoff error. We later discuss modular arithmetic which provides an efficient approach for carrying out exact arithmetic by using integers of arbitrary precision in a minimum number of operations. It is advisable to use double precision floating point numbers to store fixed-size integers, since they provide 53 bits of storage instead of the 32 bits of standard integers. Moreover, double arithmetic operations are faster on most modern computer architectures.

Exact arithmetic does not exclude floating-point inputs. In this case we need to convert the floating-point data to integer values by multiplying by an appropriate power of 10. If the input parameters contain more than 32 bits we have to use larger precision integers. Again, most of the computation is carried out over fixed-precision integers.

## 2.2 Perturbations

Geometric problems are defined in terms of maps associating any given input instance to a unique output instance.

**Definition 1** A *problem mapping* is a mapping $\Pi : X \to Y$ between topological spaces. The *input space* $X = \mathbb{R}^{nd}$ has the standard euclidean topology. The *output space* $Y$ is, generally, the product $D \times R$ of a finite space $D$ with the discrete topology and the direct union $R$ of real spaces with the euclidean topology.

A recurrent example will be the Convex Hull Volume (CHV) problem, which maps point sets to the real number expressing the volume of their convex hull. The output space is $\mathbb{R}$ with the euclidean topology, and the mapping is continuous.

Computational geometry is concerned with the effective computation of problem mappings. Often, however, the implementation of algorithms is impeded by certain conditions imposed by the algorithm designer on the input. Typically, "special" cases such as those where the mapping is discontinuous are assumed not to occur. To illustrate, consider the planar Convex Hull Face-Structure (CHF) problem where, given a point-set, the sequence of hull edges must be constructed.

The output topology is the direct union of real euclidean topologies, each corresponding to a distinct combinatorial structure of the polygon. A variety of algorithms, including Beneath-Beyond, assume that three points are never collinear. This configuration represents a discontinuity in this map because it is arbitrarily close to two input instances which give rise to outputs in disjoint components and, hence, at infinite distance. Perturbations supply a mechanism to allow programs to run and produce meaningful output even if they cannot handle these special configurations.

**Definition 2** For any input $x \in X$, a *perturbed instance* of $x$ is a curve $x()$ rooted at $x$, i.e., the image of a continuous function $x() : \mathbb{R}_{\geq 0} \rightarrow X$ such that $x(0) = x$. A *perturbation scheme* $Q$ defines a perturbed instance for every element of $X$.

For the sake of simplicity, we do not explicitly show the dependence of $x()$ on the choice of $Q$. The intuitive notion of perturbations as very small changes to the input is formalized in

**Definition 3** Given a problem mapping $\Pi : X \rightarrow Y$ and a perturbation scheme $Q$, the *perturbed problem mapping* $\overline{\Pi}$ is a mapping from $X$ to $Y$ such that

$$\overline{\Pi}(x) = \lim_{\epsilon \rightarrow 0^+} \Pi(x(\epsilon)),$$

assuming that every such limit exists.

Again, an explicit indication of the dependence of the derived mapping on the perturbation scheme is foregone.

The goal is that the new problem mapping be defined and continuous on a proper superset of the original domain, thus incorporating some or all of the special instances. We also hope that implementing an algorithm for $\overline{\Pi}$ will be easier than explicitly handling all special cases for which some given algorithm for $\Pi$ is undefined. In short, we shall solve $\overline{\Pi}$ instead of $\Pi$ and then argue that the output of $\overline{\Pi}$ can yield enough information to recover the output of $\Pi$ at the same input. The latter constitutes the *postprocessing* phase, which is in general nontrivial. However, there is a restricted yet important case in which postprocessing is superfluous:

**Proposition 4** For any perturbation scheme, if mapping $\Pi$ is continuous at $x \in X$, then $\Pi(x) = \overline{\Pi}(x)$.

This is the case with CHV, discussed in detail in Section 7. Things are less favorable for the planar CHF problem: given a point set containing subsets of more than two collinear points on the boundary, the output on perturbed input will contain edges split into more than one segments. Postprocessing then has to merge these segments by eliminating points in the interior of polygon edges. This process is analyzed for the general CHF problem in Section 7 and a practical approach is discussed for the three-dimensional CHF problem. Postprocessing for the problem of polytope intersection was examined in [8].

### 2.3 Computing with Perturbations

Given a program $\Phi$ that implements $\Pi$, the question is how to obtain another real RAM program $\overline{\Phi}$ that implements $\overline{\Pi}$. For this, $\Pi$ must be modified to be able to run on perturbed instances $x(\epsilon)$ and follow the same branches as if $\epsilon$ had an arbitrarily small positive real value. We list the syntactic changes on $\Phi$ and prove below that the new program correctly implements the perturbed map.

First, all arithmetic operations in $\Phi$ are transformed in order to handle perturbation curves. Memory locations in $\overline{\Phi}$ hold univariate functions in $\epsilon$ and a postprocessing stage eliminates $\epsilon$ from

the output by some limiting process. The latter depends on the particular problem but we assume that it is possible. Lastly, every branching operation of $\Phi$ is transformed to a branching operation that tests the limit of the sign of some $\epsilon$-function: if $f$ is the respective function tested by $\Phi$, then the new branch depends on

$$\lim_{\epsilon \to 0^+} \text{sign} f(x(\epsilon)),$$

where $\text{sign}(\cdot)$ is a piecewise constant function with values in $\{-1, 0, +1\}$.

Problematic instances always include those where $\Pi$ is discontinuous. In addition, a program may be undefined on other instances, for example two covertical points in the case of a planar CHF solved by a sweep-line algorithm. All inputs not dealt with by a program can be modeled by the vanishing of some polynomial in the input. Conforming to the standard viewpoint in the literature [2, 3, 1] we have

**Definition 5** An input instance is *degenerate* with respect to some program if and only if it causes some numerator or denominator polynomial $f$ at a branch to vanish, where $f$ is not identically zero. Equivalently, an input instance is *generic* with respect to this program if there is no such branch polynomial.

Yap [3] distinguishes between *problem-dependent* degeneracies, i.e., those where $\Pi$ is discontinuous, and *algorithm-induced* degeneracies, such as the covertical points for the sweep-line algorithm.

**Definition 6** A perturbation scheme $Q$ is *valid* with respect to a function $f$ if and only if, for every input $x \in X$, the limit

$$\lim_{\epsilon \to 0^+} \text{sign} f(x(\epsilon))$$

exists and is nonzero. Perturbation $Q$ is valid with respect to a set of functions if and only if it is valid with respect to every function in this set. $Q$ is valid with respect to a given real RAM program if and only if it is valid with respect to the set of all branch polynomials in the program.

Clearly, under a valid perturbation no degenerate inputs arise, which implies that the zero branches in a program can be ignored.

**Theorem 7** Assume that $Q$ is a valid perturbation scheme for a real RAM program $\Phi$ computing mapping $\Pi$ and that $\overline{\Phi}$ is obtained by the transformation at the beginning of this section. Then (i) $\overline{\Phi}$ computes the perturbed mapping $\overline{\Pi}$ and (ii) for $x \in X$ such that $\Pi$ is continuous, $\overline{\Phi}$ yields $\Pi(x)$. Statement (ii) holds if some, or all, of the zero branches of $\Phi$ are removed.

**Proof** (i) By validity all limits exist, hence $\overline{\Phi}$ follows the same execution path on $x(\epsilon)$ as $\Phi$ would if $\epsilon$ were specialized to an arbitrarily small real positive value. In constructing $\overline{\Phi}$ we have also assumed that postprocessing is possible. Then the map $\overline{\Pi}$ is computed by $\overline{\Phi}$. (ii) Proposition 4 establishes the second assertion. By validity no zero branches are taken in $\overline{\Phi}$, therefore these branches might as well be pruned away. □

From this theorem, it is clear that the action of perturbations can be thought of as concentrated at the branches. The main advantage of the perturbation method is that some or all of the zero branches do not need to be implemented. This brings us to the original problem stated at the beginning of Section 2.2. We now see how algorithms designed under the hypothesis of non-degeneracy can be used for solving the perturbed problem mapping, from which postprocessing can produce the output of $\Pi$.

It must be underlined that whenever a given program $\Phi$ is transformed to $\overline{\Phi}$ to reflect the application of some perturbation, all instructions should be changed according to the chosen scheme. It leads to severe inconsistencies to allow some instructions to be executed as if the perturbation were not implemented and, similarly, it is a grave error to try to use more than one scheme at a time. Imagine, for example, that in the planar CHF problem coordinate comparisons are not transformed under the perturbation, but the Orientation primitive is transformed. Then three covertical points may be detected to be so by coordinate comparisons, though for the Orientation test they are not even collinear.

So far we have formalized the notion of valid perturbations as a tool for coping with degenerate inputs but no concrete guidelines have been presented for their implementation. In later sections we examine practical ways for establishing validity, propose valid schemes covering some common geometric primitives and study the issue of efficiently executing a transformed program.

# 3 Previous work

The simplest approach in coping with degeneracies is to handle each special case separately, which is tedious for implementors and unattractive for theoreticians, though some recent work re-examines this common belief [8, 9].

Dantzig's [10] symmetry breaking rules in Linear Programming are regarded as the precursor of current systematic perturbations. The idea is to perturb the right-hand side of every constrain by an infinitesimal quantity that depends on the index of this constrain. The $i$-th constrain then becomes

$$\sum_{j=1}^{n} a_{i,j}x_j + x_{n+i} = b_i(\epsilon) = b_i + \epsilon^i,$$

where $x_1, \ldots, x_n$ are the original variables, each $x_i$ for $i > n$ is a slack variable and the $a_{i,j}$ and $b_i$ are constants.

Edelsbrunner and Mücke generalize in [2] a technique called Simulation of Simplicity (SoS for short), already presented in [11], which refines the above method. Every input coordinate $x_{i,j}$ is perturbed into

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon^{2^{i\delta - j}},$$

where $\delta > d$ and $d$ is the dimension. The perturbation is infinitesimal due to symbolic variable $\epsilon$; it is also conceptual in the sense that the computation remains numeric. Raising $\epsilon$ to such a high power intuitively distinguishes between any two coordinates, which allows SoS to be applied to a wide range of geometric primitives, including the determinantal ones examined in this paper. One exception is an inconsistency in the case of the InSphere primitive discussed in Section 6.3.

The main drawback of SoS is that it incurs an overhead to the algebraic complexity of the algorithm which is exponential in $d$ in the worst case. Specifically, deciding the sign of a $d \times d$ perturbed determinant, although rather fast on the average, requires the calculation of $\Omega(2^d)$ minors in the worst case. We indicate how this bound is established for the case of the Orientation primitive. In [2] the Orientation primitive on perturbed input is reduced to the evaluation of a sequence of minors of the original $d \times d$ matrix. Each minor that may have to be evaluated is associated with a vector of the form $(v_1, \ldots, v_{d-2})$, where $v_i \in \{1, \ldots, d\}$ and $i < j \Rightarrow v_i \leq v_j$. A simple counting argument now implies the lower bound. For other primitives a similar argument works.

Yap [3] deals with the more general setting in which branching occurs at arbitrary rational expressions and proposes a method which is equivalent to an infinitesimal perturbation [12]. Recently, it has been extended to analytic test functions [5]. For input variables $\mathbf{x} = (x_1, \ldots, x_N)$,

the scheme considers a total ordering on all power products of the form

$$w \;=\; \prod_{i=1}^{N} x_i^{e_i}, \qquad e_i \geq 0.$$

This ordering, denoted by $\leq_A$, is *admissible* if, for all power products $w, w', w''$,

$$1 \leq_A w \qquad \text{and} \qquad w \leq_A w' \Rightarrow ww'' \leq_A w'w''.$$

If $w_1 \leq_A w_2 \leq_A \ldots$ is an admissible ordering of power products larger than one, then each polynomial $f(\mathbf{x})$ at the numerator or denominator of a branch expression is associated with the infinite list

$$S(f) \;=\; (f, f_{w_1}, f_{w_2}, ...)$$

where $f_{w_k}$ is the partial derivative of $f$ with respect to $w_k$. The sign of $f$ is taken to be the sign of the first polynomial in $S(f)$ with a nonzero value, which can always be found after a finite number of evaluations. The worst case complexity to evaluate this new test is exponential in $N$, although the average case complexity is significantly lower. Consider sparse $N$-variate polynomials with degree in each variable bounded by $m$. If all variables are of the same maximum degree then $f$ has at least $m^N$ partial derivatives, and if all of them must be evaluated then the algebraic complexity is $\Omega(m^N)$. When the input consists of $n$ $d$-vectors the complexity of evaluating the determinantal tests of this article is exponential in $d$.

Each power product $w$ is uniquely defined by the vector $e = (e_1, \ldots, e_N)$ of exponents. Defining the admissible ordering reduces to a fixed ordering of the product vectors of some $N \times N$ matrix with each vector $e$. Choosing the matrix essentially picks the direction of perturbation, and in this sense this scheme gives a *controlled* perturbation. Since we are free in choosing this matrix, it is possible to simulate either SoS or any of our schemes presented below.

Very recently, we were informed by Michelucci of his own perturbation scheme, applicable to arbitrary rational expressions [13]. It is reminiscent of Yap's but allows some control on the direction in which points are perturbed as well as branching functions on derived quantities. Its drawbacks are the high computational overhead and the need of language capabilities such as streams in order to avoid creating a formidable implementation task.

Dobrindt, Mehlhorn and Yvinec proposed an efficient scheme specifically for coping with degenerate intersections between a convex and a general polyhedron in three dimensions [8]. Since the vertices of the convex polyhedron are guaranteed to be perturbed in a specific direction with respect to the given facets, this is another instance of a controlled perturbation. Another merit of this work is that it discusses postprocessing in detail in order to recover the exact solution.

A recent account of postprocessing and its implementation difficulties for convex hulls in general dimension can be found in [14]. This paper also formalizes the use of modular arithmetic as an efficient way of implementing exact arithmetic, and shows how randomization can further reduce computational cost. This method is briefly sketched in section 5.

Knuth [15] provides a nice treatment of the planar case, setting the problem of Delaunay triangulations and CHF in combinatorial terms, which are related to oriented matroids. The examined primitives are the two-dimensional restrictions of Ordering and InSphere. The proposed mechanism for avoiding coincident, collinear and cocircular point sets is equivalent to an infinitesimal perturbation, reminiscent of SoS.

A *structural* perturbation for a motion-planning algorithm, in which the input objects are the semi-algebraic sets describing the obstacles, is given by Canny in [16]. He uses towers of infinitesimals to eliminate degeneracies while preserving essential properties of the sets, namely emptiness and cardinality of connected components.

Emiris and Canny presented in [1] an infinitesimal perturbation that constitutes the first efficient scheme, inspired by the SoS method. For geometric algorithms, every coordinate $x_{i,j}$ is deterministically perturbed into

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon\, i^j, \tag{1}$$

where $\epsilon$ is a symbolic infinitesimal. The perturbation applies to the Orientation and Transversality primitives with worst case complexity overheads $\mathcal{O}(\log d)$ and $\mathcal{O}(d^{1+\alpha})$ under the algebraic and bit models respectively and requires no symbolic computation; $\alpha$ is an arbitrarily small positive constant. These bounds are obtained by assuming that the unperturbed program implements sign determination by determinant evaluation and that a constant fraction of input points are distinct.

For a wider class of algorithms with branching at arbitrary rational expressions randomization is traded for efficiency [1]. The $i$-th perturbed point is

$$x_i(\epsilon) = x_i + \epsilon r_i, \qquad 1 \leq i \leq n,$$

where $r_i$ is a random integer uniformly chosen from a sufficiently large interval $R \subset \mathbb{Z}$. Let $D$ denote the maximum degree in the input variables of any polynomial in the program, let $\phi(n, s)$ be the program's bit complexity and $s$ an upper bound on the size of the input coordinates. Then the probability that the scheme fails to eliminate some degeneracy is bounded by $D\tau/\#R$, where $\#R$ denotes the cardinality of $R$ and $\tau < \phi(n, s)$ is the number of branch polynomials. The approach here is of the Las Vegas type because the fact that some degeneracy is not removed can be detected deterministically in which case the program is restarted. The algebraic complexity overhead is $\mathcal{O}(D^{1+\alpha})$ and the worst case bit complexity overhead is $\mathcal{O}(\phi^{2+\alpha}(n, s))$, for an arbitrarily small $\alpha > 0$ accounting for the polylogarithmic factor.

# 4 Linear Perturbations

The efficiency of scheme (1) is essentially due to the linearity of the $\epsilon$-factor. This section weakens the requirements on validity for linear perturbations and provides a powerful validity criterion for a specific class of linear schemes.

*Linear* perturbations are of the following type:

$$x_i(\epsilon) = x_i + \epsilon b_i, \qquad 1 \leq i \leq n, \tag{2}$$

where $x_i = (x_{i,1}, \ldots, x_{i,d})$ and $b_i = (b_{i,1}, \ldots, b_{i,d}) \in \mathbb{R}^d$ are the $i$-th input and perturbation vectors respectively. Let $f(x_1, \ldots, x_n)$ be any polynomial in $n$ vector variables; its *initial form* is a homogeneous polynomial $\mathcal{I}(f)$ in the same variables, equal to the sum of all terms in $f$ of maximum total degree. For homogeneous polynomials $f = \mathcal{I}(f)$.

**Theorem 8** Let $g(x_1, \ldots, x_n) = \mathcal{I}(f)$ be the initial form of $f$. For linear perturbation (2) to be valid with respect to polynomial $f$, it suffices that $g(b_1, \ldots, b_n) \neq 0$.

**Proof** Consider $f(x(\epsilon))$ as a univariate polynomial in $\epsilon$. From Definition 6, it is required that $f(\epsilon)$ never vanishes on perturbed input. If at least one coefficient is never zero, the polynomial is not identically zero and its zero set does not include all real numbers. The highest-order coefficient in $f(\epsilon)$ is $g(b_1, \ldots, b_n) \neq 0$, therefore the zero set of $f(\epsilon)$ is not fully dimensional which implies that $f(\epsilon)$ has a finite number of roots. It suffices now to assume that $\epsilon$ takes real values smaller than the minimum positive root of $f(\epsilon)$. □

The significance of this theorem is twofold. First, the validity requirement has to be tested only against the initial form of the branch polynomial. More importantly, the problem of designing an efficient perturbation scheme is reduced to finding a single set of input vectors $b_1, \ldots, b_n$ on which the initial forms of all branch polynomials do not vanish. In practice, one may use known point sets such as points on the moment curve, employed by scheme (1), or those defined by the rows of a Cauchy matrix, which would provide an alternative to scheme (1). In general, though, defining perturbation vectors $b_1, \ldots, b_n$ is a hard problem, a stronger version, in fact, of the zero avoidance problem [17].

This theorem can be generalized to nonlinear perturbations.

### 4.1 A Validity Criterion

We establish a rather general criterion for showing validity. This criterion was motivated by the application of scheme (1) to the InSphere primitive. This primitive decides, given points $x_{i_1}, \ldots, x_{i_{d+2}} \in \mathbb{R}^d$, whether $x_{i_{d+2}}$ lies in the interior of the hypersphere defined by the first $d+1$ points. InSphere is shown in Section 6.3 to reduce to testing the sign of a $(d+2)\times(d+2)$ determinant therefore validity follows by the nonsingularity of matrix

$$W_{d+2} = \begin{bmatrix} 1 & i_1 & \ldots & i_1^d & \sum_{j=1}^d i_1^{2j} \\ 1 & i_2 & \ldots & i_2^d & \sum_{j=1}^d i_2^{2j} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & i_{d+2} & \ldots & i_{d+2}^d & \sum_{j=1}^d i_{d+2}^{2j} \end{bmatrix}.$$

The proof requires Descartes' rule of sign which we state here for completeness. We consider a univariate polynomial $u_1 + u_2 x + \cdots + u_N x^{N+1}$ in *canonical* form, i.e., $u_i$ is the nonzero coefficient of the $i$-th smallest power of the variable. The number of sign variations of the polynomial's nonzero coefficients $u_1, \ldots, u_N$ is the number of consecutive pairs $(u_k, u_{k+1})$, $1 \leq k < N$, such that the product $u_k u_{k+1}$ is negative.

**Proposition 9 (Descartes' Rule of Sign) [18]** The number of sign variations of a polynomial's nonzero coefficients exceeds the number of positive zeros, multiplicities counted, by an even nonnegative integer.

**Proposition 10** Matrix $W_{d+2}$ is nonsingular for distinct positive $i_j$, $1 \leq j \leq d+2$.

**Proof** If $W_{d+2}$ is singular then there is a nonzero vector $(q_1, \ldots, q_{d+2})$ in the kernel of the matrix. Therefore the $y$-polynomial $\sum_{j=0}^d q_{j+1} y^j + q_{d+2} \sum_{j=1}^d y^{2j}$ has at least $d+2$ distinct positive zeros, namely $i_1, \ldots, i_{d+2}$. The polynomial has also at most $d+1$ sign variations, which contradicts Descartes' rule. □

Here we generalize the discussion to include potentially more primitives in addition to InSphere. The perturbation is restricted to the form:

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon \beta_i^{\gamma_j}, \qquad 1 \leq i \leq n, 1 \leq j \leq d, \tag{3}$$

with $\gamma = (\gamma_1, \ldots, \gamma_d) \in \mathbb{Z}^d$ fixed and $b_i = (\beta_i^{\gamma_1}, \ldots, \beta_i^{\gamma_d})$ as the $i$-th perturbation vector, where $\beta_i \in \mathbb{R}$ for $1 \leq i \leq n$. For a general polynomial $f(w_1, \ldots, w_t)$, the *support* of $f$, denoted $\mathrm{supp}(f)$, is the set of integer exponent vectors that correspond to nonzero coefficients:

$$A = \mathrm{supp}(f) \subset \mathbb{Z}^t \iff f = \sum_{a \in A} c_a w^a, \quad c_a \neq 0$$

where, for exponent vector $a = (a_1, \ldots, a_t) \in \mathbb{Z}^t$, we write $w^a = \prod_{k=1}^{t} w_k^{a_k}$.

To generalize the class of primitives examined, we let the entries of the determinant be arbitrary $d$-variate polynomials $f_j$. Consider $t$ polynomials each on $t$ sets of variables, $f_j(x_i) = f_j(x_{i,1}, \ldots, x_{i,d})$, where $t \leq n$, $1 \leq i, j \leq t$. Let $A_j = \mathrm{supp}(f_j) \subset \mathbb{Z}^d$. If # denotes cardinality, let the union of all singleton supports be $U = \bigcup_{\#A_j=1} A_j$. Now define

$$B_j = A_j \setminus U \ \subset A_j \qquad 1 \leq j \leq t.$$

Hence $B_j = \emptyset$ if $\#A_j = 1$. Let the set of inner products of exponent vectors for each $B_j$ with a fixed vector $\gamma = (\gamma_1, \ldots, \gamma_d) \in \mathbb{Z}^d$ be

$$C_j = \{\langle \gamma, a \rangle \in \mathbb{Z} \,|\, a \in B_j\}, \qquad 1 \leq j \leq t.$$

Each $C_j$ has a minimum and maximum element denoted $\min C_j$ and $\max C_j$ respectively. We restrict attention to primitives expressed as determinants of order $t$, with the $(k, j)$-th entry equal to $f_j(x_k)$. By Theorem 8 we have to examine the determinant that has $f_j(b_k)$ as the $(k, j)$-th entry.

**Theorem 11** Suppose that $\beta_i$ is positive and distinct for every $1 \leq i \leq n$ in the perturbation scheme (3). Moreover each polynomial $f_j$ has coefficients of the same sign and, possibly after reindexing, the nonempty sets $C_j$ are ordered so that, for every $j > 1$, $\max C_{j-1} < \min C_j \leq \max C_j < \min C_{j+1}$. Then the $t \times t$ matrix with $(i, j)$-th entry $f_j(b_i)$ is nonsingular.

The proof of the theorem relies on the technical lemma below. Intuitively, we consider $t$ polynomials that correspond to the columns of the matrix. The hypothesis of the theorem separates the truncated supports, i.e., the subsets obtained after ignoring all elements in singleton supports.

**Technical Lemma 12** If $q_1, \ldots, q_t$ are any real values, $y$ is a real variable, $\gamma \in \mathbb{Z}^d$ is fixed and polynomials $f_j$ satisfy the hypothesis of Theorem 11, then polynomial $F(y) = \sum_{j=1}^{t} q_j f_j(y^{\gamma_1}, \ldots, y^{\gamma_d})$ has less than $t$ positive real roots.

**Proof** By expanding $f_j(y^{\gamma_1}, \ldots, y^{\gamma_d}) = \sum_{a \in A_j} c_{j,a} y^{\langle \gamma, a \rangle}$, the univariate polynomial $F(y)$ can be written

$$F(y) = \sum_{j=1}^{t} q_j \sum_{a \in A_j} c_{j,a} y^{\langle \gamma, a \rangle} = \sum_{j=1}^{t} \sum_{a \in A_j} q_j c_{j,a} \, y^{\langle \gamma, a \rangle} = \sum_{j : B_j \neq \emptyset} \sum_{a \in B_j} q_j c_{j,a} \, y^{\langle \gamma, a \rangle} + \sum_{j : \#A_j = 1} \sum_{a \in A_j} q'_j c_{j,a} \, y^{\langle \gamma, a \rangle}.$$

The expression relies on the fact that the supports $A_j$ are nonempty and are partitioned between non-singletons in the first summand and singletons in the second. Coefficients $q'_j$ in the second summand result from the contribution of $a \in U \cap A_j$ for which $B_j \neq \emptyset$, in other words $a \in A_j \setminus B_j \subset U$. After manipulating the first summand we obtain

$$F(y) = \sum_{a \in \cup B_j} \left( \sum_{j : a \in B_j} q_j c_{j,a} \right) y^{\langle \gamma, a \rangle} + \sum_{j : \#A_j = 1} \sum_{a \in A_j} q'_j c_{j,a} \, y^{\langle \gamma, a \rangle}.$$

Now partition the first summand into sums of monomials whose exponents belong in a certain $C_j$. Since all $B_j$ are distinct and by hypothesis the $C_j$ are ordered, the coefficient of $y^{\langle \gamma, a \rangle}$ is a single product $q_j c_{j,a}$. Furthermore, there is no sign variation among the coefficients of each sum because all $c_{j,a}$ for a fixed $j$ have the same sign. Hence, in the first summand the number of distinct coefficients is at most the number of nonempty $B_j$. This implies that the number of distinct coefficient signs is also bounded by the number of nonempty $B_j$.

In the second summand the total number of coefficients is at most equal to the number of singleton supports. Therefore, there exist at most $t$ distinct coefficient signs in $F(y)$, hence the number of sign variations is at most $t-1$. An application of Descartes' rule completes the proof. □

**Proof of Theorem 11** If the matrix is singular, then there must exist a nonzero real vector $(q_1, \ldots, q_t)$ in the kernel of the linear transformation of the matrix, therefore the univariate polynomial $F(y)$ from Lemma 12 has a distinct positive root $\beta_i$ for all $1 \leq i \leq t$. The existence of $t$ distinct positive roots contradicts Lemma 12. □

For the InSphere primitive and matrix $W_{d+2}$, $t = d+2$, $f_1 = 1$, $f_j(b_k) = i_k^j$ for $2 \leq j \leq d+1$ and $f_{d+2}(b_k) = \sum_{l=1}^{d} i_k^{2l}$ where $1 \leq k \leq d+2$. Then Proposition 10 follows as a corollary.

# 5 Evaluation of Branch Expressions

Some algebraic techniques are presented for the efficient evaluation of primitives in perturbed programs. An important consequence of these techniques is that no computation in the derived program need involve the infinitesimal variable. Thus, although the perturbation is symbolic, all arithmetic is numeric.

We first discuss interpolation as a general method for computing univariate polynomials in $\epsilon$ from their values. The only assumption here is that the total degree $\delta$ of each polynomial is known. The first step is to obtain a sequence of interpolation pairs. These are pairs of $\epsilon$ specializations, usually at distinct primes, and the respective values of the polynomial. If $\delta+1$ interpolation pairs are available, *dense interpolation* can be used to compute the coefficients in $\mathcal{O}(\delta \log^2 \delta)$ arithmetic operations [7].

If, furthermore, there is an *a priori* bound $T$ on the number of nonzero terms that is significantly lower than the maximum number $\delta+1$, then *sparse interpolation* is preferable. There exists a probabilistic algorithm with algebraic complexity $\mathcal{O}(\delta\tau^{1+\alpha})$ that requires $\mathcal{O}(\delta\tau)$ interpolation pairs, where $\tau \leq T$ is the actual number of nonzero terms in the polynomial and $\alpha$ is any positive constant. A deterministic algorithm has complexity $\mathcal{O}(T^{2+\alpha} \log \delta)$ and requires $2T$ interpolation pairs, where $\alpha$ again accounts for the polylogarithmic factor. Both algorithms are surveyed in [17].

Traditionally, the cost of evaluating the unknown polynomial is of minor concern in the context of the interpolation problem, yet here this cost must be assessed. In general, computing one interpolation pair takes time proportional to the complexity of evaluating the polynomial. For the important case of determinantal tests, i.e., tests expressed as determinants, the complexity of the evaluation phase dominates the overall complexity. The rest of this section concentrates on determinantal tests of order $t$ and develops more efficient ways for the combined problem of evaluation and sign determination.

Let $MM(t) = \mathcal{O}(t^{2.376})$ [19] be the algebraic complexity of multiplying two $t \times t$ matrices and $I$ (resp. $I_t$) the identity matrix (of order $t$). The total cost of dense interpolation is $\mathcal{O}(\delta MM(t))$, where $\delta \geq t$. An improved technique for interpolating determinants whose entries are higher-degree polynomials in several variables appears in [20]. Here we present a near-optimal technique for the case in which the entries are univariate polynomials.

Given $t \times t$ matrix $A(\epsilon)$ with polynomial entries in a single variable $\epsilon$, we can express it as a matrix polynomial

$$A(\epsilon) = A_r\epsilon^r + A_{r-1}\epsilon^{r-1} + \cdots + A_1\epsilon + A_0 \tag{4}$$

where $r$ is the maximum degree in $\epsilon$ of any matrix entry. If $A_r$ is singular the approach of [21] can

be applied; in the current context $A_r$ is always nonsingular. Then we premultiply both sides by $A_r^{-1}$ to obtain

$$A_r^{-1}A(\epsilon) = I\epsilon^r + A_r^{-1}A_{r-1}\epsilon^{r-1} + \cdots + A_r^{-1}A_1\epsilon + A_r^{-1}A_0.$$

The determinant of the right-hand side equals [22] the characteristic polynomial of

$$C = \begin{bmatrix} 0 & I_t & 0 & \cdots & 0 \\ 0 & 0 & I_t & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & I_t \\ -A_r^{-1}A_0 & -A_r^{-1}A_1 & -A_r^{-1}A_2 & \cdots & -A_r^{-1}A_{r-1} \end{bmatrix}.$$

This discussion reduces the sign determination of a univariate matrix to a matrix inversion and a characteristic polynomial computation.

**Theorem 13** Let $A(\epsilon)$ be a matrix of order $t$, whose entries are linear univariate polynomials in $\epsilon$; then $A(\epsilon) = A_1\epsilon + A_0$, where $A_0$ and $A_1$ are numeric matrices. If $A_1$ is nonsingular, determining the sign of $\det(A(\epsilon))$ can be reduced to computing determinant $\det A_1$ and the characteristic polynomial of matrix $-A_1^{-1}A_0$.

**Proof** By the nonsingularity of $A_1$ we can write $A(\epsilon)$ as

$$A(\epsilon) = (-A_1)\left(-I\epsilon - A_1^{-1}A_0\right).$$

If we represent sign with a value in $\{-1, 0, +1\}$, then the sign of $\det A(\epsilon)$ is the product of the signs of $\det A_1$ and $\det(-I\epsilon - A_1^{-1}A_0)$ multiplied by $(-1)^t$. The last determinant is simply the characteristic polynomial of $-A_1^{-1}A_0$. □

A discussion of modular arithmetic is in order here because, in addition to being a common method for conducting arithmetic on computers, it is also the fastest with respect to bit complexity for evaluating the perturbed tests. Recall that the perturbation method requires exact computation; modular arithmetic allows us to carry most of the computation over fixed-precision integers. For this we may use double precision floating point numbers which provide 53 bits for storing an integer. Besides the classical application of modular arithmetic to integer arithmetic, it can be used with rational data with the same asymptotic complexity [23].

The basic approach is as follows. First the given quantities are mapped to their residues modulo a set of primes, then the required computation is performed within each finite field defined by every one of these primes and, lastly, the true answer is computed by the results in each finite field. The last step relies on the Chinese Remainder Theorem. In order for the entire process to be deterministic, a bound on the value of the final answer must be known. This is used to calculate the number of different finite fields.

Let $k$ denote the number of finite fields $\mathbb{Z}_p$, for distinct primes $p$, necessary to carry out a particular computation. The first and third stage have each bit complexity $\mathcal{O}(M(k)\log k)$. The middle stage is the actual computation within each $\mathbb{Z}_p$ and its bit complexity is $k$ times the algebraic complexity of this computation. All primes $p$ have constant bit size, independent of the size of the answer. We have made the implicit assumption that a sufficiently long list of primes is available at the beginning of the computation and that obtaining $p$ from the list is a constant-time operation. Both hypotheses are easy to guarantee by using some upper bound on the input size and by storing an array of primes.

In practice, randomization is traded for efficiency in determining the number $k$ of finite fields in a dynamic fashion. Namely, at the end of every finite field computation, we reconstruct the answer with respect to the information gathered so far. When few primes have been used, the reconstructed answer changes with every new residue. But once enough finite fields have been used the true answer is obtained and, of course, does not change. When the reconstructed answer is stable for two or three consecutive finite fields we may assume, with very high probability, that the true answer has been calculated. Typically this happens well before the worst case number of finite fields has been used.

**Corollary 14** The algebraic complexity of computing $\det(A_1\epsilon + A_0)$ is $\mathcal{O}(MM(t)\log t)$, where $t$ is the order of matrices $A_0$ and $A_1$. Let $s$ be the maximum bit size of any entry in $A_1$ and $A_0$. Then the bit complexity of computing the above determinant is $\mathcal{O}((ts)^{1+\alpha} + tsMM(t)\log t)$, for some arbitrarily small positive constant $\alpha$.

**Proof** The operations required are a matrix inversion, a matrix multiplication, calculation of a determinant and computation of the coefficients of a characteristic polynomial. Each takes $\mathcal{O}(MM(t))$ time, except from the last step which takes $\mathcal{O}(MM(t)\log t)$ time, for arbitrary matrices, due to an algorithm by Keller-Gehrig [24]. To establish the bit complexity bound, the transformation of Theorem 13 is used. The bit size of the coefficients of the $\epsilon$-polynomial representing the determinant is $\mathcal{O}(ts)$ since the original matrix entries have size $s$ and its order is $t$. Hence modular arithmetic may be used over $k = \mathcal{O}(ts)$ fields. □

# 6 Some Common Primitives

This section deals with specific perturbation schemes for certain common determinantal primitives, namely with extending the application of

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon\, i^j \tag{1}$$

to Ordering and InSphere and with a more efficient variant that optimizes the bit size of the perturbation quantities:

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon(i^j \bmod q), \qquad 1 \le i \le n, 1 \le j \le d, \tag{5}$$

where $q$ is the smallest prime that exceeds $n$. The bit size of the perturbation quantities is bounded by $\log n$, which is optimal, since there must be at least $n$ distinct such quantities.

We make two assumptions in estimating the complexity of the original program. First, that sign determination of determinants is implemented by a determinant calculation. If some cheaper way was employed to find out the sign [25] the overhead would increase roughly by a linear factor in $d$. Second, we assume that the number of distinct input parameters is a constant fraction of $n$. Hence, if $s$ denotes an upper bound on the bit size of the input data, $s = \Omega(\log n)$.

We place the emphasis on designing efficient valid perturbations from a computational complexity point of view. The comparison on complexities is carried out between the worst case complexities of programs and their perturbed counterparts. This encourages the design of schemes efficient for the almost-generic cases, as opposed to very special cases. This is not an artifact of our schemes but rather an inherent property of the perturbation method: by reducing arbitrary instances to the generic treatment, special characteristics cannot be exploited. As a consequence, the perturbed output and hence the complexity of the perturbed program are not particularly sensitive to the complexity of the exact answer. For instance, given $n$ coincident points as input to an algorithm solving the CHF problem, the exact output is a single point whereas the perturbed output is a polytope with $n^{\lfloor d/2 \rfloor}$ facets. See [9] for a discussion of limitations of the perturbation approach.

### 6.1 Ordering

The Ordering, or Sorting, primitive decides the order of two quantities expressing the $k$-th coordinate of the $i_1$-th and $i_2$-th input points. On input perturbed with the perturbation defined by (1) the primitive decides the sign of

$$x_{i_1,k}(\epsilon) - x_{i_2,k}(\epsilon) \;=\; x_{i_1,k} + \epsilon\, i_1^k - x_{i_2,k} - \epsilon\, i_2^k.$$

For degenerate inputs the factors of the infinitesimal must be compared, which comes down to comparing $i_1$ against $i_2$. Notice that this is the lexicographic ordering. Since all indices are distinct, the perturbation is valid by Theorem 8. The perturbation of Equation (5), for $k = 1$, is valid too.

The evaluation requires in the worst case, an extra constant-time check. Under the bit model, the extra comparison adds a $\mathcal{O}(\log n)$ term, which is upper bounded by the original bit complexity because each $x_{i,j}$ is assumed to be $\Omega(\log n)$ bits long.

**Theorem 15** The perturbation defined by (1) is valid with respect to the Ordering primitive and does not change the asymptotic running-time complexity of this primitive in the algebraic as well as the bit model. The same holds for the perturbation of Equation (5) for comparisons along the first coordinate.

If we compare along some general $k$-th coordinate, validity may not hold for the second scheme.

### 6.2 Orientation and Transversality

Given a query point $x_{i_{d+1}}$ and a hyperplane in $\mathbb{R}^d$ spanned by points $x_{i_1}, \ldots x_{i_d}$, Orientation, or Sidedness, decides in which one of the two halfspaces defined by this hyperplane the query point lies. A degeneracy occurs exactly when $x_{i_{d+1}}$ lies on the hyperplane. The primitive is formulated as a test of a determinant sign; the relevant matrix is $\Lambda_{d+1}$ below. Transversality determines the orientation of $d$ points in $\mathbb{R}^{d-1}$ given by their homogeneous coordinates and is expressed as the sign of $\det(\Delta_d)$, where

$$\Lambda_{d+1} = \begin{bmatrix} 1 & x_{i_1,1} & x_{i_1,2} & \ldots & x_{i_1,d} \\ 1 & x_{i_2,1} & x_{i_2,2} & \ldots & x_{i_2,d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{i_{d+1},1} & x_{i_{d+1},2} & \ldots & x_{i_{d+1},d} \end{bmatrix}, \qquad \Delta_d = \begin{bmatrix} x_{i_1,1} & x_{i_1,2} & \ldots & x_{i_1,d} \\ x_{i_2,1} & x_{i_2,2} & \ldots & x_{i_2,d} \\ \vdots & \vdots & & \vdots \\ x_{i_d,1} & x_{i_d,2} & \ldots & x_{i_d,d} \end{bmatrix}.$$

Matrix $\Delta_d$ also comes up in a dual context, when the input objects are hyperplanes in $(d-1)$-dimensional space and Transversality decides on which side of the first hyperplane lies the intersection of the other $d-1$ hyperplanes.

For completeness we state the following proposition which relies on Theorem 8, the properties of Vandermonde matrices and Corollary 14.

**Proposition 16** [1] Perturbation (1) is valid with respect to algorithms that branch on determinants of $\Lambda_{\delta+1}$ and $\Delta_\delta$, for $\delta \leq d$, where $d$ is the space dimension. The perturbation increases the asymptotic running-time complexity of evaluating the primitive, under the algebraic model, by $\mathcal{O}(\log d)$. Under the bit model, the worst case complexity is increased by a factor of $\mathcal{O}(d^{1+\alpha})$, where $\alpha$ is an arbitrarily small positive constant.

Now consider scheme (5). By Theorem 8 the nonsingularity of $\Lambda_{d+1}(\epsilon)$ is obtained by using the closed form expression of a Vandermonde determinant.

$$\det \overline{V_{d+1}} = \det \begin{bmatrix} 1 & i_1 \bmod q & \dots & i_1^d \bmod q \\ 1 & i_2 \bmod q & \dots & i_2^d \bmod q \\ \vdots & \vdots & & \vdots \\ 1 & i_{d+1} \bmod q & \dots & i_{d+1}^d \bmod q \end{bmatrix} \equiv \prod_{k>l\geq 1}^{d+1} (i_k - i_l) \not\equiv 0 \pmod q.$$

Validity in the case of Transversality follows similarly. The crucial property for both schemes is that they define $n$ vectors, every $d$ of which are linearly independent.

The sign of $det\Lambda_{d+1}(\epsilon)$ and $det\Delta_{d+1}(\epsilon)$ is the sign of the least significant term in the respective polynomial. One way to compute it, adopted by SoS, is to calculate directly all terms, starting with the one of least degree, until finding one that does not vanish. Fortunately, our scheme lends itself to the more efficient technique of Theorem 13. The first column of $\Lambda_{d+1}(\epsilon)$ has to be multiplied by $\epsilon$, then both perturbed matrices satisfy the theorem's hypothesis.

**Theorem 17** Perturbation (5) is valid with respect to Orientation and Transversality. It increases the worst case algebraic and bit complexities of the Orientation and Transversality primitives by a $\mathcal{O}(\log d)$ factor.

**Proof** Validity follows from Theorem 8. The original algebraic complexity is $\Theta(MM(d))$ [26]. From Corollary 14, the complexity on perturbed input is $\mathcal{O}(MM(d)\log d)$. The original worst case bit complexity depends on the size of the answer which is $\Theta(ds)$. Typically modular arithmetic is used, requiring $\Theta(ds)$ different finite fields, while on perturbed input the number of finite fields is $\mathcal{O}(d(s+\log n))$. The assumption that $s = \Omega(\log n)$ finishes the proof. □

An important feature for implementors is that the growth of any computed quantity is quasi-linear in the dimension. For instance, in a 3-dimensional problem with input quantities of absolute magnitude less than $10^5$, any computed quantity fits in a computer word.

### 6.3 InSphere

We apply (1) to the InSphere primitive which generalizes to arbitrary dimension the planar InCircle primitive. InSphere decides, given $d+2$ points, whether the $(d+2)$-nd point lies in the interior of the higher-dimensional sphere defined by the first $d+1$ points in $\mathbb{R}^d$. It can be reduced to testing the sign of a determinant as follows. First, lift all points to the surface of a paraboloid in $\mathbb{R}^{d+1}$ by adding a $(d+1)$-st coordinate equal to the sum of the squares of the $d$ coordinates defining each point. The original space is a $d$-dimensional hyperplane which the paraboloid touches at the origin. Let $x_{i_1}, x_{i_2}, \dots, x_{i_{d+1}}$ be the points defining the sphere; their lifted images define a hyperplane $H$ in $\mathbb{R}^{d+1}$. The query point $x_{i_{d+2}}$ lies within the sphere if and only if its lifted image lies below $H$, in other words to the same side of $H$ as the original points. A degeneracy occurs exactly when $x_{i_{d+2}}$ lies on the sphere or, equivalently, on $H$, which happens exactly at the singularities of

$$, _{d+2} = \begin{bmatrix} 1 & x_{i_1,1} & x_{i_1,2} & \dots & x_{i_1,d} & \sum_{j=1}^d x_{i_1,j}^2 \\ 1 & x_{i_2,1} & x_{i_2,2} & \dots & x_{i_2,d} & \sum_{j=1}^d x_{i_2,j}^2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & x_{i_{d+1},1} & x_{i_{d+1},2} & \dots & x_{i_{d+1},d} & \sum_{j=1}^d x_{i_{d+1},j}^2 \\ 1 & x_{i_{d+2},1} & x_{i_{d+2},2} & \dots & x_{i_{d+2},d} & \sum_{j=1}^d x_{i_{d+2},j}^2 \end{bmatrix}.$$

Eliminating degeneracies for the particular matrix could be achieved by the "cheap trick" of [2] which perturbs the points on the higher-dimensional paraboloid, by perturbing the sum of squares as if it were an additional coordinate. However, this may lead to inconsistencies in some special configurations, if the same algorithm also uses another primitive such as $\Lambda_{d+1}$. Consider, for instance, deciding the relative position of a line and a circle which touch at two coincident points $x_1$ and $x_2$, by using the Orientation primitive on the line and $x_1$ and the InSphere primitive on the circle and $x_2$.

Validity reduces by Theorem 8 to proving the nonsingularity of the Vandermonde-resembling matrix

$$W_{d+2} = \begin{bmatrix} 1 & i_1 & \dots & i_1^d & \sum_{j=1}^d i_1^{2j} \\ 1 & i_2 & \dots & i_2^d & \sum_{j=1}^d i_2^{2j} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & i_{d+2} & \dots & i_{d+2}^d & \sum_{j=1}^d i_{d+2}^{2j} \end{bmatrix},$$

which follows from Proposition 10 or Theorem 11. Unfortunately, the hypothesis of the theorem is not readily satisfied by the second perturbation (5). A similar scheme, with residues taken mod $q$, with $q = \Omega(n^{d-1})$, has been recently shown [27] to be valid, offering a slight improvement on complexity.

For the first scheme, the perturbed determinant expands to

$$det,_{\,d+2}(\epsilon) = \begin{vmatrix} 1 & x_{i_1,1}(\epsilon) & \dots & x_{i_1,d}(\epsilon) & \epsilon^2 \sum_{j=1}^d i_1^{2j} + \epsilon(2\sum_{j=1}^d x_{i_1,j} i_1^j - i_1^{d+2}) \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & x_{i_{d+2},1}(\epsilon) & \dots & x_{i_{d+2},d}(\epsilon) & \epsilon^2 \sum_{j=1}^d i_{d+2}^{2j} + \epsilon(2\sum_{j=1}^d x_{i_{d+2},j} i_{d+2}^j - i_{d+2}^{d+2}) \end{vmatrix} +$$

$$+ \begin{vmatrix} 1 & x_{i_1,1}(\epsilon) & \dots & x_{i_1,d}(\epsilon) & \sum_{j=1}^d x_{i_1,j}^2 + \epsilon i_1^{d+2} \\ & \vdots & & \vdots & \vdots \\ 1 & x_{i_{d+2},1}(\epsilon) & \dots & x_{i_{d+2},d}(\epsilon) & \sum_{j=1}^d x_{i_{d+2},j}^2 + \epsilon i_{d+2}^{d+2} \end{vmatrix}. \qquad (6)$$

Computing each of the two determinants can be reduced to a characteristic polynomial computation. For the first determinant, it suffices to move an $\epsilon$ factor from the last to the first column, which reduces the problem to the sign determination of

$$\det \begin{bmatrix} \epsilon & x_{i_1,1}(\epsilon) & \dots & x_{i_1,d}(\epsilon) & \epsilon \sum_{j=1}^d i_1^{2j} + (2\sum_{j=1}^d x_{i_1,j} i_1^j) - i_1^{d+2} \\ \vdots & \vdots & & \vdots & \vdots \\ \epsilon & x_{i_{d+2},1}(\epsilon) & \dots & x_{i_{d+2},d}(\epsilon) & \epsilon \sum_{j=1}^d i_{i_{d+2},j} + (2\sum_{j=1}^d x_{i_{d+2},j} i_{d+2}^j) - i_{d+2}^{d+2} \end{bmatrix}.$$

The entries of this matrix are linear in $\epsilon$ and the leading matrix coefficient is $W_{d+2}$ above, so Theorem 13 is applied.

For the sign of determinant (6), the first column is multiplied by $\epsilon$ and Theorem 13 is again applied. Now the leading matrix coefficient is simply a Vandermonde matrix of order $d+2$.

**Theorem 18** The perturbation of Equation (1) is valid with respect to the InSphere primitive and increases its algebraic complexity by a $\mathcal{O}(\log d)$ factor. Under the bit model, the worst case complexity increases by a $\mathcal{O}(d^{1+\alpha})$ factor, where $\alpha$ is an arbitrarily small positive constant.

**Proof** Validity is already established, based on Theorem 11. The original algebraic complexity is $\Theta(MM(d))$ and, by Corollary 14, the new complexity is $\mathcal{O}(MM(d)\log d)$. In the worst case, the determinant has size $\Theta(ds)$. With modular arithmetic the original bit complexity is then

$$\Theta(d^2(ds)^{1+\alpha} + dsMM(d)),$$

where $\alpha$ denotes the smallest of several positive constants.

For the perturbed primitive, modular arithmetic is used and the number of finite fields is $\mathcal{O}(ds + d^2 \log n)$ since this is the coefficient size in each of the two characteristic polynomials that must be computed. This bound follows from the fact that each coefficient is the sum of certain minors of a matrix of order $d + 2$, whose entries have bit size bounded by $\max\{s, d \log n\}$. Hence the bit complexity of evaluating the primitive is

$$\mathcal{O}(d^2(ds + d^2 \log n)^{1+\alpha} \; + \; (ds + d^2 \log n) MM(d) \log d).$$

The overhead now follows from $s = \Omega(\log n)$. □

### 6.4 Limitations

Primitives that decide on the relative position of derived objects may pose a limitation to our method. Consider, for instance, the two-dimensional ham-sandwich algorithm in [28], where lines are the input objects and their intersection points are the derived objects. The three primitives of the algorithm are: deciding whether a point lies above or below a line; comparing the first coordinate of two points; and comparing the distances of two points from a line.

Applying scheme (1) to the points removes all degeneracies but it is not clear that this does not create some inconsistent configuration. Applied to the input lines, SoS successfully perturbs them into general position. However, perturbation (1) is valid only for the first test. Consider a scheme using the first $n$ primes, denoted as $q_1, \ldots, q_n$:

$$x_{i,j}(\epsilon) \; = \; x_{i,j} \; + \; \epsilon(q_i^j). \tag{7}$$

The bit complexity of the perturbation quantities is then $\mathcal{O}(d \log n)$. Applied to the ham-sandwich algorithm, the perturbation of Equation (7) is valid for the first and second tests but not for the third one. One should keep in mind that consistency requires that exactly one scheme is applied to all primitives of a specific algorithm.

Another problem involving derived objects is the line-segment intersection problem. Given a finite set of line segments in the plane, the goal is to construct the planar arrangement defined by the segments and their intersections. The hardest primitive here is to decide on which halfplane, with respect to a query segment, the intersection of two segments lies.

## 7 Computing Convex Hulls

Convex hull computation in general dimension is a fundamental geometric problem with a wide variety of applications, such as visibility and illumination in geometric modeling and graphics [29, 30], predicting the poses of industrial parts on a conveyor belt and computing stable grasps [31, 32], collision detection in robotics and animation [33], material identification in geology [34], molecular docking in drug fabrication [35] as well as in solving systems of nonlinear equations [36, 37, 21].

This section discusses our implementation of the Beneath-Beyond algorithm [11] which solves the Convex Hull Volume (CHV) problem for finite input sets of integral points in arbitrary dimension, and reports on the running-time performance. The implementation produces approximate solutions to the Convex Hull Face-Structure (CHF) problem, in a sense specified later. We examine the issue of postprocessing that arises when we wish to recover the exact facet structure. The section concludes with an application to finding a stable grasp of a robot.

The algorithm is designed under the assumption of non-degeneracy. The perturbation of Equation (5) is applied in order to allow arbitrary inputs, since the only two tests needed are Ordering

Table 1: Performance of the Convex Hull Volume implementation.

| | | user CPU running time | | |
|---|---|---|---|---|
| $d$ | $n$ | $\rho \simeq 1$ (random) | $\rho \simeq .5$ | $\rho = 0$ (coincident) |
| 4 | 54 | 4s | | 24s |
| 4 | 100 | 8s | 40s | 1m 6s |
| 4 | 200 | 19s | 1m 11s | 2m 8s |
| 4 | 500 | 53s | | 7m 39s |
| 3 | 100 | 0s | | 11s |
| 4 | 100 | 8s | 40s | 1m 6s |
| 5 | 100 | 43s | 4m 7s | 5m 19s |
| 6 | 100 | 4m 18s | | 45m 15s |
| 7 | 100 | 29m 1s | | |

on the first coordinate and Orientation. The exact volume of the convex hull is possible to obtain without any postprocessing because, by Proposition 4, the problem mapping

$$\mathrm{CHV} : \mathbb{Z}^{dn} \to \mathbb{Q}$$

is continuous everywhere. The algorithm sorts all points on their first coordinate and then proceeds incrementally by adding each new point to the convex hull of all previous points. Due to the perturbation, each region between the new point and the existing hull can be partitioned into $d$-simplices, each defined by the new point and one of the visible facets. Then, it is straightforward to compute the exact volume by summing all simplex volumes whose expression as an $\epsilon$-polynomial has a nonzero constant term.

The *extreme* points of a given point-set are those that strictly maximize the inner product with some $d$-vector, i.e., they are not expressible as a convex combination of the other points; these are exactly the vertices of the convex hull. Perturbation (5) guarantees that the output polytope is simplicial; its vertex set is a superset of the extreme points because it may contain some points that are not extreme but simply *extremal*, i.e., they maximize the inner product with a certain vector. Also, the number of facets may not be minimum because of the extremal points reported as vertices and because all facets are triangulated.

If a specific application required that the output polytope had the minimum number of facets regardless of whether they are simplices or not, then certain adjacent facets would have to be merged. This can be accomplished by comparing the normals of every two adjacent facets. The normal of a facet can be computed in $\mathcal{O}(MM(d))$ and there are $d$ tests per facet, hence this postprocessing does not affect the asymptotic complexity of the program. A more detailed account of postprocessing for convex hulls and its implementation can be found in [14].

The implementation has benefited from code written by H. Rosenberger, E. Mücke and D. Manocha. The current version is in Ansi-C and includes about 1000 lines of code for the main combinatorial part, 600 lines for the perturbation part and 1400 lines for the modular and exact integer arithmetic package. It is free for distribution either by anonymous `ftp` from `robotics.eecs.Berkeley.edu /pub/ConvexHull` or at `http://www.inria.fr/safir/SAFIR/Ioannis-eng.html`.

Table 7 shows the performance of the program on a SparcStation 10/41 with one 40 MHz processor, 32 MBytes of memory and a rating of 50 SpecInt92. Each Orientation test comprises of a heuristic calculation of $\det \Lambda_{d+1}$; only if this vanishes is the reduction to a characteristic polynomial

undertaken. As before, $d$ and $n$ stand for the dimension and the number of input points respectively and all coordinates are integers in $(-100, 100)$. The output of the program is the rational volume and a list of facets, each described by the defining input points; no postprocessing was implemented. The user CPU running times are rounded down to an integer number of seconds.

For fixed $d$ and $n$ we have experimented on inputs of various degrees of degeneracy. The last three columns are headed by the approximate fraction $\rho$ of Orientation tests whose evaluation is nonzero on the original input; these tests are carried out as determinant calculations. Thus, the first column corresponds to random inputs with practically all tests being generic. The other extreme has inputs with coincident points, constructed to test the program's performance when all Orientation tests reduce to a characteristic polynomial. The middle column corresponds to point sets comprised of random points, generated as above, and coincident points, at an appropriate ratio.

In analyzing these results it must be remembered that the program's complexity depends on the number of facets in the partial convex hulls. In the worst case, the hull of $n$ points in $d$ dimensions has $\mathcal{O}(n^{\lfloor d/2 \rfloor})$ facets. However, the expected number of facets for points selected randomly as above is proportional to $\log^{d-1} n$ [38]; this is verified by our experimental results.

The first author has adapted this program for commercial use in computing the pose of an industrial part that falls on a conveyor belt. The algorithm computes the convex hull facets, then projects them on a sphere to compute the probability of the object landing on this facet [31]. This information is then used by the gripper in order to grasp and move the part from the belt. The input comes in floating-point numbers: we calculate the smallest power of 10 that produces signed integers of at most 31 bits and multiply the data by this power. For this application the precision is always sufficient. More importantly, the three-dimensional hulls must have all coplanar facets merged, no degenerate facets with zero area and no non-extreme vertices.

We choose to maintain the necessary planarity information as we build the hull, instead of computing normals and comparing them. This is easy to do because of the way the Orientation primitive has been implemented: if $\det \Lambda_4$ vanishes, then we know that only because of the perturbation the query point appears to be off the given plane. Hence we can mark the appropriate new facets as being on the same plane as the old facet. Every two-dimensional facet carries a positive integer that serves as an index to the plane that contains it. One-dimensional facets carry two negative integers corresponding to the two planes delimiting the hull which intersect at this facet. Zero-dimensional facets are not encountered due to a preprocessing that has eliminated all duplicate sites. Once the three-dimensional polytope is built, coplanar facets are merged and, for each plane, a two-dimensional problem is solved to clear the non-extreme points.

Maintaining the coplanarity information, merging facets and running the two-dimensional subproblems requires an additional 500 lines of code. It is not clear that in arbitrary dimension this approach to postprocessing would be optimal and, moreover, whether the perturbation method would be the best alternative when the facet structure is the final goal. In general, when the complexity of postprocessing becomes significant, the value of the perturbation technique should be reexamined against alternative strategies.

# 8 Conclusion

We have defined the notion of input perturbation and have concentrated on linear schemes, which are amenable to efficient computation techniques. In particular, we have proposed two such schemes that are valid for certain important geometric primitives, including InSphere. The merit of these schemes is twofold. First, their simplicity makes them attractive for practical use and, second, they

are the most efficient to date.

A research direction is to develop optimal schemes applicable to the widest possible class of primitives. It is also interesting to examine whether it is possible, in general, to control the direction at which the input points are perturbed; this would simplify postprocessing.

The basic existential question on the perturbation method is still open. After the flurry of papers proposing different perturbation schemes, recent work argues against the general applicability of this method [8, 9, 39] motivated by the observation that the difficulty and complexity of postprocessing might sometimes dominate that of the entire program. Indeed, perturbations should not be treated as a panacea but should rather be considered in the specific context of the algorithm and the output desired.

# References


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

[2] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics*, 9(1):67–104, 1990.

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

[4] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. ACM Symp. on Computational Geometry*, pages 74–82, 1992.

[5] R. Seidel. The nature and meaning of perturbations in geometric computing. In *Proc. 11th Symp. Theoret. Aspects Computer Science, Lect. Notes Computer Science 775*, pages 3–17. Springer-Verlag, 1994.

[6] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[7] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.

[8] K. Dobrindt, K. Mehlhorn, and M. Yvinec. A complete framework for the intersection of a general polyhedron with a convex one. In *Proc. 3rd Workshop Algorithms Data Struct., Lect. Notes Comp. Science 709*, pages 314–324. Springer-Verlag, Berlin, 1993.

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

[10] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, 1963.

[11] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, 1987.

[12] C.-K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comp. Sys. Sci.*, 40:2–18, 1990.

[13] D. Michelucci. An epsilon-arithmetic for removing degeneracies. In *IEEE Symp. Computer Arithmetic*, Bath, July 1995.

[14] I.Z. Emiris. A complete implementation for computing general dimensional convex hulls. Technical Report 2551, INRIA Sophia-Antipolis, France, 1995. Also submitted for publication.

[15] D.E. Knuth. *Axioms and Hulls*, volume 606 of *Lect. Notes in Comp. Science.* Springer-Verlag, Berlin, 1992.

[16] J.F. Canny. Computing roadmaps of semi-algebraic sets. In *Proc. Intern. Symp. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, New Orleans, 1991.

[17] R. Zippel. Interpolating polynomials from their values. *J. Symbolic Computation*, 9:375–403, 1990.

[18] G.E. Collins and R. Loos. Real zeros of polynomials. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 83–94. Springer-Verlag, Wien, 2nd edition, 1982.

[19] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9:251–280, 1990.

[20] D. Manocha and J. Canny. Multipolynomial resultants and linear algebra. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, pages 96–102, 1992.

[21] I.Z. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, December 1994.

[22] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

[23] J.H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra*. Academic Press, London, 1988.

[24] W. Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theor. Comp. Sci.*, 36:309–317, 1985.

[25] K.L. Clarkson. Safe and effective determinant evaluation. In *Proc. IEEE Symp. Foundations of Comp. Sci.*, pages 387–395, 1992.

[26] J. von zur Gathen. Algebraic complexity theory. In J. Traub, editor, *Annual Review of Computer Science*, pages 317–347. Annual Reviews, Palo Alto, Cal., 1988.

[27] T. Thiele, 1993. Personal Communication.

[28] H. Edelsbrunner and R. Waupotitsch. Computing a ham-sandwich cut in two dimensions. *J. Symbolic Comput.*, 2:171–178, 1986.

[29] S. Teller and C.H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–69, 1991.

[30] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. Partitioning and Ordering Large Radiosity Computations. *Computer Graphics*, 28, 1994.

[31] J. Wiegley, A. Rao, and K. Goldberg. Computing a statistical distribution of stable poses for a polyhedron. In *Proc. 30th Annual Allerton Conf. on Comm. Control and Computing*, U.Ill. Urbana-Champaign, 1992.

[32] K. Goldberg and M.T. Mason. Bayesian grasping. In *Proc. IEEE Conf. Robotics and Automation*, 1990.

[33] M.C. Lin, D. Manocha, and J. Canny. Efficient contact determination for dynamic environments. In *Proc. IEEE Conf. Robotics and Automation*, pages 602–608, 1994.

[34] J.W. Boardman. Automated spectral unmixing of Aviris data using convex geometry concepts. In *Proc. 4th Airborne Geoscience Workshop*, Washington, D.C., October 1993.

[35] M.L. Connolly. Molecular Interstitial Skeleton. *Computers Chem.*, 15(1):37–45, 1991.

[36] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. *Math. Comp.*, 64(212):1542–1555, 1995.

[37] I. Emiris and J. Canny. A practical method for the sparse resultant. In M. Bronstein, editor, *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, pages 183–192, Kiev, July 1993.

[38] J.L. Bentley, H.T. Kung, M. Schkolnick, and C.D. Thompson. On the average number of maxima in a set of vectors. *J. ACM*, 25:536–543, 1978.

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