Efficient Perturbations for Handling Geometric Degeneracies[1]

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

June 6, 1994

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 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 asymptotic complexity. The deterministic scheme of Emiris and Canny [1] was the first efficient scheme and is extended, in a consistent manner, to cover a wide class of geometric algorithms. We introduce a variant scheme, applicable to a restricted class of algorithms, which is almost optimal in terms of the algebraic as well as the 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 are reported.

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 formalization of perturbations based on their syntactic definition as curves rooted at input instances. A limiting process is employed to define perturbations, thus conforming to the intuitive notion of infinitesimal change. We also discuss how to recover the answer to the original problem from the output on perturbed input, either directly or after some case-specific postprocessing. Lastly, 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 first efficient scheme in the sense that the algebraic complexity overhead is at most logarithmic in the dimension. The second contribution of this article is the design and application of specific efficient schemes to a

[1] Some preliminary results have appeared in: I. Emiris and J. Canny, An Efficient Approach to Removing Geometric Degeneracies, *Proc. 8th ACM Symp. Comput. Geom.*, 1992, pp. 74–82.

[2] Computer Science Division, University of California, Berkeley CA 94720, USA; {emiris,jfc,seidel}@cs.Berkeley.edu. The first two authors supported by a David and Lucile Packard Foundation Fellowship and by NSF Presidential Young Investigator Grant IRI-8958577. The third author supported by NSF Presidential Young Investigator Grant CCR-9058440.

family of primitives. We extend the previous scheme so that it applies to four basic primitives, most notably InSphere, and propose a new variant for Orientation, Transversality and Ordering that reduces the bit-complexity overhead also 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 computed grow in a quasi-linear fashion with the dimension.

Furthermore, these schemes are simple to implement; to illustrate this claim, our third contribution is an implementation of the Beneath-Beyond algorithm [4] 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. Convex hull computation in general dimension is a fundamental geometric problem with a wide variety of applications, such as visibility and illumination in modeling and graphics [5, 6], collision detection in robotics and animation [7], material identification in geology [8], molecular docking in drug fabrication [9] as well as in solving systems of nonlinear equations, particularly in modeling, robotics and vision [10, 11].

This article 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 important primitives in Section 6, where the complexity claims are demonstrated. Section 7 presents our implementation of the Beneath-Beyond algorithm. The conclusion summarizes the main results and suggests some open questions.

2 Definitions

2.1 Computational model

Our model is the real Random Access Machine (RAM) of [12]. 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, called Ordering, is the comparison of coordinates with branch polynomial $f = x_{ij} - x_{kj}$ 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}) 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 $O(b)$, addition and subtraction, multiplication and division and, lastly, the Greatest Common Divisor (GCD) operation require respectively $O(b)$, $O(b \log b \log\log b)$ and $O(b \log^2 b \log\log b)$ bit operations [13]. We shall use $M(b) = O(b \log^2 b \log\log b)$ as an upper bound on the bit complexity of any arithmetic or GCD operation between any two rational numbers with numerator and denominator of size $O(b)$. Then 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.

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 running 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 $\mathbf{R}$ 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 $x() : \mathbb{R}_{\geq 0} \to X$ 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 $\Pi : X \to Y$ and a perturbation scheme Q, the *perturbed problem mapping* $\overline{\Pi}$ is a mapping from X to Y such that

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

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 Π is undefined. In short, we shall solve $\overline{\Pi}$ instead of Π and then argue that the output of $\overline{\Pi}$ can yield enough information to recover the output of Π 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 Π 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 hull boundary, the output polygon 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. Postprocessing for the problem of polytope intersection is examined in [14].

2.3 Computing with Perturbations

Given a program Φ that implements Π, the question is how to obtain another real RAM program $\overline{\Phi}$ that implements $\overline{\Pi}$. First, all arithmetic operations in Φ are transformed in order to handle perturbation curves. Memory locations in $\overline{\Phi}$ hold univariate functions in ϵ and a postprocessing stage eliminates ϵ from the output. Lastly, every branching operation of Φ is transformed to a branching operation that tests the limit of the sign of some ϵ-function, namely

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

if f is the respective function tested by Φ; sign() is a piecewise constant function with values in $\{-, 0, +\}$.

Problematic instances always include those where Π is discontinuous. In addition to these, 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. Alternatively, an input instance is *generic* with respect to this program if there is no such branch polynomial.

Some authors distinguish between *problem-dependent* degeneracies i.e. those where Π 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 Φ computing mapping Π and that $\overline{\Phi}$ is obtained by the transformation at the beginning of this section. Then $\overline{\Phi}$ computes the perturbed mapping $\overline{\Pi}$ and, for $x \in X$ such that Π is continuous, $\overline{\Phi}$ yields $\Pi(x)$. The statement holds if some, or all, of the zero branches of Φ are removed.

Proof By validity all limits exist, hence the map $\overline{\Pi}$ is well-defined and computed by $\overline{\Phi}$. Proposition 4 establishes the second assertion. Since, by validity, no zero branches are taken in $\overline{\Phi}$, 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 Π.

It must be underlined that whenever a given program Φ 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 [14, 15].

Dantzig's [16] symmetry breaking rules in Linear Programming are regarded as the precursor of current systematic perturbations. The principal idea is to perturb the right-hand side of every constraint equation by an infinitesimal quantity that depends on the index of this equation. The i-th constraint 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 [17], 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 ϵ; it is also conceptual in the sense that the computation remains numeric. Raising ϵ 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. Its main drawback is that it incurs an overhead to the algebraic complexity of the algorithm which is exponential in d in the worst case: 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. To prove the latter bound for the case of the Orientation primitive it suffices to count all vectors $(v_1, \ldots, v_{d-2})$ such that d is the order of the matrix, $v_i \in \{1, \ldots, d\}$ and $i < j \Rightarrow v_i \leq v_j$. In [2] every such vector is associated with a minor that may have to be evaluated.

Yap in [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, as proven in [18]. Recently, it has been extended to analytic test functions [19]. 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 $w_1, w_2, \ldots$ 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

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

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 non-zero 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 $\Omega(m^N)$.

Dobrindt, Mehlhorn and Yvinec proposed an efficient scheme specifically for coping with degenerate intersections between a convex and a general polyhedron in three dimensions [14]. It is noteworthy that the vertices of the convex polyhedron are guaranteed to be perturbed in a specific direction with respect to the given facets. Another merit of this work is that it discusses postprocessing in detail in order to recover the exact solution.

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 [20]. He uses towers of infinitesimals to eliminate degeneracies while preserving essential properties of the sets, namely emptiness and number 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 ϵ is a symbolic infinitesimal. The perturbation applies to the Orientation and Transversality primitives with worst-case complexity overheads $O(\log d)$ and $O(d^{1+\alpha})$ under the algebraic and bit models respectively and requires no symbolic computation; α is an arbitrarily small positive constant. Notice that all perturbations relevant to geometric algorithms satisfy the validity conditions set in Definition 6.

For a wider class of algorithms with branching at arbitrary rational expressions randomization is traded for efficiency and 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 $R \subset \mathbf{Z}$. Let D denote the maximum degree in the input variables of any polynomial in the program, $\phi(n, s)$ 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 is the length of R and $\tau < \phi(n, s)$ is the number of branch polynomials. The approach here is of the Last Vegas variety 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 $O(D^{1+\alpha})$ and the worst-case bit complexity overhead is $O(\phi^{2+\alpha}(n, s))$.

4 Linear Perturbations

The efficiency of scheme (1) is essentially due to the linearity of the ϵ-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 and multiplication is scalar. 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, as is typically the case in geometric algorithms, $f = \mathcal{I}(f)$.

Theorem 8 Let $g(x_1, \ldots, x_n) = \mathcal{I}(f)$ be the initial form of f. For a 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 ϵ. 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 ϵ 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 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 [21]. This theorem can be readily generalized to nonlinear perturbations.

4.1 A Validity Criterion

This criterion was motivated by the application of scheme (1) to the InSphere primitive which 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. This primitive is shown in Section 6.3 to reduce to testing the sign of a $(d+2) \times (d+2)$ determinant which implies that validity rests, by Theorem 8, upon 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. Given a univariate polynomial in canonical form, the number of sign variations of its 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) [22] The number of sign variations of a polynomial's nonzero coefficients exceeds the number of positive zeros, multiplicities counted, by an even non-negative integer.

Proposition 10 Matrix W_{d+2} is non-singular for distinct positive i_j, $1 \le j \le d+2$.

Proof If W_{d+2} is singular 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:

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

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 $1 \le i \le n$. 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:

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

where, for exponent vector $a = (a_1, \ldots, a_t) \in \mathbb{Z}^t$, we write $w^a = \prod_{k=1}^{t} w_k^{a_k}$. Consider t polynomials $f_j(x_i) = f_j(x_{i,1}, \ldots, x_{i,d})$, where $t \le n$, $1 \le i, j \le t$. Let $A_j = \mathrm{supp}(f_j) \subset \mathbb{Z}^d$, let the union of all singleton supports be $U = \bigcup_{\#A_j = 1} A_j$, where $\#$ denotes cardinality, and define

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

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

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

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 (k,j)-th entry equal to $f_j(b_k)$.

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

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 fewer 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

$$F(y) = \sum_{j=1}^{t} q_j \sum_{a \in A_j} c_{j,a} y^{\langle \gamma, a \rangle} = \sum_{j=1}^{t} \sum_{a \in A_j} q_j c_{j,a}\, y^{\langle \gamma, a \rangle} = \sum_{j : B_j \neq \emptyset} \sum_{a \in B_j} q_j c_{j,a}\, y^{\langle \gamma, a \rangle} + \sum_{j : \#A_j = 1} \sum_{a \in A_j} q_j c_{j,a}\, y^{\langle \gamma, a \rangle}$$

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

The expression relies on the fact that the supports A_j are non-empty and are partitioned between non-singletons in the first summand and singletons in the second.

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. 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 β_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 ϵ from their values. The only assumption here is that the total degree of each polynomial is known, call it δ. The first step is to obtain a sequence of interpolation pairs, in other words pairs of ϵ 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 $O(\delta \log^2 \delta)$ arithmetic operations [13].

If, furthermore, there is an *a priori* bound T on the number of non-zero terms that is significantly lower than the maximum number $\delta+1$, then *sparse interpolation* is preferable. There exists a probabilistic algorithm with algebraic complexity $O(\delta\tau^{1+\alpha})$ that requires $O(\delta\tau)$ interpolation pairs, where $\tau \leq T$ is the actual number of non-zero terms in the polynomial and α is any positive constant. A deterministic algorithm has complexity $O(T^{2+\alpha} \log \delta)$ and requires $2T$ interpolation pairs, where α again accounts for the polylogarithmic factor. Both algorithms are surveyed in [21].

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 the interpolation pairs takes time proportional to the number of pairs times the complexity of evaluating the polynomial. For the important case of determinantal tests discussed in this article, 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 $MM(t) = O(t^{2.376})$ [23] be the algebraic complexity of multiplying two $t \times t$ matrices and I (I_t) the identity matrix (of order t). Dense interpolation costs $O(\delta MM(t))$, where $\delta \geq t$. An improved technique for interpolating determinants whose entries are higher-degree polynomials in several variables appears in [24]. Here, following [1], we generalize a near-optimal technique for the case when the entries are univariate polynomials.

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

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

where r is the maximum degree in ϵ of any matrix entry. If A_r is non-singular, we have

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

and the determinant of the right-hand side equals [25] the characteristic polynomial of

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

If A_r is singular there exist $rt \times rt$ matrices P and Q such that $\det(A(\epsilon)) = \det(P\epsilon + Q)$. Moreover, [24] demonstrates a series of transformations that produces matrices P' and Q' of smaller order, such that the original problem is reduced to computing $\det(P'\epsilon + Q')$; the worst-case complexity of this method is $O(rtMM(rt)\log(rt))$. The primitives of this article always fall within the first case. Furthermore, r is typically a small constant, therefore this approach is almost optimal in the sense that it incurs only a polylogarithmic overhead on the asymptotic complexity.

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

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. Besides the classical application to integer arithmetic [13], modular methods and the Chinese Remainder Theorem can be used with rational data with the same asymptotic complexity [26].

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. This 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, which is used to calculate the number of different finite fields used.

Let k denote the number of finite fields $\mathbf{Z}_p$, for distinct primes p, necessary to carry out a particular computation, where k depends on the bit size of the final answer. The first stage of mapping each given quantity to its respective k residues, as well as the third stage of calculating the answer from its k residues, have each bit complexity $O(M(k)\log k)$. The middle stage is the actual computation within each $\mathbf{Z}_p$ and its bit complexity is k times the algebraic complexity of this computation. We have made the implicit assumption that each prime p has constant bit size and that choosing such a prime, from an existing and sufficiently long list, is a constant-time operation.

Corollary 14 The algebraic complexity of computing $\det(A_1\epsilon + A_0)$ is $O(MM(t)\log t)$, 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 $O((ts)^{1+\alpha} + tsMM(t)\log t)$, for some arbitrarily small positive constant α.

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 $O(MM(t))$ time, except from the last step which takes $O(MM(t)\log t)$ time, for arbitrary matrices, due to an algorithm by Keller-Gehrig [27]. To establish the bit complexity bound, the transformation of Theorem 13 is used. The bit size of the coefficients of the ϵ-polynomial representing the determinant is $O(ts)$ since the original matrix entries have size s and its order is t. Hence modular arithmetic may be used over $k = O(ts)$ 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:

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon(i^j \bmod q), \qquad 1 \le i \le n, 1 \le j \le d, \tag{4}$$

where ϵ denotes the symbolic infinitesimal and 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. It is reasonable to suppose that at least a constant fraction of the n input vectors are distinct. Hence, if s denotes an upper bound on the bit size of the input data, $s \ge \log n$.

In practice, the emphasis is on designing efficient valid perturbations from a computational complexity point of view. The comparison is carried out between 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. The reason is that the overhead incurred by perturbing is not output-sensitive. 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 [15] for a discussion of limitations of this approach.

For avoiding very degenerate cases like this and for ensuring the lower bound on s a preprocessing phase can be used to detect and eliminate duplicates without affecting the asymptotic complexity of most algorithms.

6.1 Ordering

This 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 (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. Perturbation (4), 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 $O(\log n)$ factor, which is upper bounded by the original bit complexity because each $x_{i,j}$ is $\Omega(\log n)$ bits long.

Theorem 15 Perturbation (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 perturbation (4) for comparisons along the first coordinate.

Unfortunately, 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 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 Λ_{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 $\det(\Delta_d)$:

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

Matrix Δ_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 as in [28], for example, where $d = 3$.

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 Δ_δ, 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 $O(\log d)$. Under the bit model, the worst-case complexity is increased by a factor of $O(d^{1+\alpha})$, where α is an arbitrarily small positive constant.

Now consider scheme (4). By Theorem 8 the non-singularity of $\Lambda_{d+1}(\epsilon)$ is obtained by using the closed form expression of a Vandermonde determinant.

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

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. Both perturbed matrices satisfy the theorem's hypothesis; for $\Lambda_{d+1}(\epsilon)$ to do so, the first column is multiplied by ϵ, which does not affect the determinant sign.

Theorem 17 Perturbation (4) 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 $O(\log d)$ factor.

Proof Validity follows from Theorem 8. The original algebraic complexity is $\Theta(MM(d))$ [29]. From Corollary 14, the complexity on perturbed input is $O(MM(d)\log d)$. 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 $O(d(s+\log n))$, The assumption that $s \geq \log n$ 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 this primitive test which 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 $x_{i_1}, x_{i_2}, \ldots, x_{i_{d+1}}$ 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

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

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 Λ_{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

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

which follows from Proposition 10 or Theorem 11. Unfortunately, the hypothesis of the theorem is not readily satisfied by the second perturbation (4). A similar scheme, with residues taken $\bmod q$, with $q = \Omega(n^{d-1})$, has been recently shown [30] to be valid, offering a slight improvement on complexity.

The perturbed determinant expands to the sum

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

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

Computing each of the two determinants can be reduced to a characteristic polynomial computation by Theorem 13. For the first determinant, it suffices to move an ϵ factor from the last to the first column, while for the second determinant the first column must be multiplied by ϵ.

Theorem 18 Perturbation (1) is valid with respect to the InSphere primitive and increases its algebraic complexity by a $O(\log d)$ factor. Under the bit model, the worst-case complexity increases by a $O(d^{1+\alpha})$ factor, where α is any 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 $O(MM(d)\log d)$. 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 α denotes the smallest of several positive constants.

For the perturbed primitive modular arithmetic is used and the number of finite fields is $O(ds + d^2 \log n)$ 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 the maximum of s, for the input data, and $d \log n$ for the perturbation quantities. Hence the bit complexity of evaluating the primitive is

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

The overhead now follows from $s \geq \log n$. □

6.4 Other Primitives

Our techniques directly apply to several other primitives including those presented in [2]. 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 [31] with lines on the plane being the input objects and their intersection points being 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) fails for the second test. We considered a scheme using the first n primes, denoted as $q_1, \ldots, q_n$:

$$x_{i,j}(\epsilon) = x_{i,j} + \epsilon(q_i^j). \tag{5}$$

The bit complexity of the perturbation quantities is then $O(d \log n)$. Applied to the ham-sandwich algorithm, perturbation (5) is valid for the second but not for the third test. One should keep in mind that consistency requires that exactly one scheme is applied to all primitives of a specific algorithm.

7 Computing Convex Hulls

This section discusses our implementation of the Beneath-Beyond algorithm [4] 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 from the output.

The algorithm is designed under the assumption of non-degeneracy; perturbation (4) is applied in order to allow arbitrary inputs, since the only two primitive tests needed are Ordering 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

$$\text{CHV} : \mathbf{Z}^{dn} \to \mathbf{Q}$$

Table 1: Performance of the Convex Hull Volume implementation.

d	n	user CPU running time $\rho \simeq 1$ (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		

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 existing facets. Then, it is straightforward to compute the exact volume by summing all simplex volumes whose expression as an ϵ-polynomial has a non-zero constant term. Note that extension to rational inputs is possible without affecting the asymptotic complexity; for the case of modular arithmetic see [26].

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 (4) 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. Hence the output vertex set is not always of minimum cardinality. 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 facets adjacent to a ridge, for every ridge. The normal of a facet can be computed in $O(MM(d))$ and there are d tests per facet, hence this postprocessing does not affect the worst-case complexity of the program.

The implementation has benefited from code written by H. Rosenberger, E. Mücke and D. Manocha. The current version is in C and free for distribution. It includes about 1000 lines for the main combinatorial part, 600 lines for the perturbation part and 1400 lines for the modular and big-integer exact arithmetic package. The performance of the program on certain input instances is reported on Table 7, for experiments on a SparcStation 10 computer with one 40 MHz processor. Each Orientation test comprises of a heuristic calculation of $\det \Lambda_{d+1}$; only if this vanishes the reduction to a characteristic polynomial is 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 ρ 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 $O(n^{\lfloor d/2 \rfloor})$ facets. However, the expected number of facets for points selected randomly as above is proportional to $\log^{d-1} n$ [32]; this is verified by our experimental results.

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. 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 schemes applicable to a wide class of primitives, including InSphere, while optimizing complexity. 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, some objections have recently been voiced against the general applicability of this method [14, 15] motivated by the observation that the difficulty and complexity of postprocessing might dominate that of the entire program.

References

[1] I.Z. Emiris and J.F. Canny. A general approach to removing degeneracies. In *Proc. 32nd IEEE Symp. Found. of Comp. Sci.*, pages 405–413, 1991. Full version to appear in *SIAM J. Comput.*

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

[5] S. Teller and C.H.Séquin. Visibility Preprocessing for Interactive Walkthroughs. *Computer Graphics (Proc. SIGGRAPH '91)*, 25(4):61–69, 1991.

[6] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. Partitioning and Ordering Large Radiosity Computations. *Computer Graphics (Proc. SIGGRAPH '94)*, 28, 1994.

[7] M.C. Lin, D. Manocha, and J. Canny. Efficient Contact Determination for Dynamic Environments. In *Proc. IEEE Intern. Conf. Robotics and Automation*, pages 602–608, 1994.

[8] J.W. Boardman. Automated Spectral Unmixing of Aviris Data Using Convex Geometry Concepts. In *Proc. 4th Airborne Geoscience Workshop*, Washington, D.C., October 1993.

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

[10] B. Huber and B. Sturmfels. A polyhedral method for solving sparse polynomial systems. To appear in Mathematics of Computation. A preliminary version presented at the "Workshop on Real Algebraic Geometry", August 1992.

[11] I. Emiris and J. Canny. A Practical Method for the Sparse Resultant. In *Proc. ACM Intern. Symp. on Symbolic and Algebr. Computation*, pages 183–192, 1993.

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

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

[14] 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.

[15] 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.

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

[17] H. Edelsbrunner. Edge-skeletons in arrangements with applications. *Algorithmica*, 1:93–109, 1986.

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

[19] 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.

[20] J.F. Canny. Computing roadmaps of semi-algebraic sets. In *Proc. Intern. Symp. Applied Algebra, Algebraic Algor. and Error-Corr. Codes*, 1991.

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

[22] 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.

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

[24] D. Manocha and J. Canny. Multipolynomial Resultants and Linear Algebra. In *Proc. ACM Intern. Symp. on Symbolic and Algebr. Computation*, pages 96–102, 1992.

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

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

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

[28] H. Edelsbrunner and L.J. Guibas. Topologically sweeping an arrangement. In *Proc. 18th Annual ACM Symp. Theory of Comp.*, pages 389–403, 1986.

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

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

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

[32] 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.