An Efficient Approach to Removing Geometric Degeneracies

Ioannis Z. Emiris *

Computer Science Division
University of California, Berkeley
Berkeley CA 94720

Abstract

We wish to increase the power of an arbitrary geometric algorithm designed for non-degenerate input, by allowing it to execute over arbitrary inputs. This paper describes a deterministic direct perturbation of the input which applies to algorithms whose branching decisions depend on determinants in the input parameters. We concentrate on four predicates that cover most important algorithms in computational geometry. The method alters the input parameters by infinitesimal amounts to guarantee that the perturbed input lies in general position. It is an attractive candidate for practical use, and is considerably simpler as well as more efficient than existing approaches. Specifically, it is the first method with a complexity overhead that is polynomial in the input size and, in most cases, a small constant. Under our real computation model, the asymptotic complexity remains unaffected; the bit complexity is at worst increased by a factor proportional to $d^{2+\alpha}$, where d is the dimension of the geometric space of the input objects and α an arbitrarily small positive constant. A variation of the perturbation, applied to two predicates, achieves optimal bit size for the perturbation quantities and is even more efficient. Lastly, we illustrate the applicability of our approach.

*Supported by a David and Lucile Packard Foundation Fellowship and by NSF Presidential Young Investigator Grant IRI-8958577.

1 Introduction

Algorithms in computational geometry are typically designed for input instances in general position. The treatment of degenerate cases is usually tedious and intricate, thus seldom straightforward. Degenerate situations are therefore excluded from the theoretical discussion, yet they remain a nontrivial matter for implementors. In this paper we describe a general approach to avoid having to deal with geometric degeneracies.

Take, for instance, the convex hull problem in d dimensions and consider your favorite algorithm for this problem. It is typically described under the hypothesis of general position which assumes that no more than d points lie on the same $(d-1)$-dimensional hyperplane. This essentially supposes that some relevant predicate, called Orientation below, can always decide the side on which the $(d+1)$st point lies with respect to the hyperplane spanned by the first d points. This assumption does not always hold in practice. Systematic methods that have been proposed introduce a slight perturbation of the input points and this is also our approach. For instance, perturbing 4 coplanar points in 3-dimensional space will result in one of the following two configurations. Either one face defined by exactly 3 of the points with the fourth one in the interior of the convex hull, or two or three adjacent faces. Both scenarios will produce a convex hull arbitrarily close to, if not the same as, the one corresponding to the unperturbed input set.

Previous systematic approaches include [9] and [17]; their main drawback is that they incur a time complexity at least as large as exponential in the space dimension. We introduce a direct deterministic perturbation of the input parameters that guarantees the systematic removal of degeneracies with respect to four common predicates that cover most geometric algorithms. The perturbation does not affect those sets of objects that already lie in general position and will always produce a consistent approximate solution, arbitrarily close to the one corresponding to the original input. Our method applies to algorithms which can be represented as *algebraic branching programs*; the running time of such an algorithm on the perturbed input is within a small constant factor of the original running time. Under the *bit model*, the overhead is polynomial and negligible for small dimension. A variation of our scheme, applied to two predicates of interest, achieves optimal bit size of the perturbation quantities and is even more efficient. Our scheme's conceptual elegance and computational efficiency makes it a competitive candidate for use by future geometric software packages.

The next section provides all definitions. Section 3 is a comparative study of previous work on handling degeneracies and Section 4 describes our direct perturbation schemes for algorithms with determinant tests. This section demonstrates the main results. In Section 5 we examine some applications and, lastly, we discuss future directions in our work.

2 Preliminaries

2.1 Computation model

We describe first a very general setting for dealing with the problem of input degeneracy. The model encompasses algorithms that perform arithmetic operations and can branch.

The *input* is a finite set of numeric values from some ordered infinite field and the *output* is a set of finite objects. We focus on the field of reals, although our results apply to any ordered infinite field, such as the rationals or any real field extension. Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$ be the input variables and $\mathbf{a} = (a_1, \ldots, a_n)$ an actual input instance. Influenced by [1] as well as [14] we define an *algebraic branching program* to be a (finite) directed graph G together with a function Φ. There exists exactly one *input vertex* with no incoming edge in G. All vertices have zero, one or three outgoing edges. Function Φ assigns

- to every vertex v of outdegree one (*operation vertex*) an instruction of the form

$$f_v = f_{v_1} \circ f_{v_2} \text{ or } f_v = c \circ f_{v_1},$$

  where $\forall\, i \in \{1,2\}$, either v_i belongs to every path from the input vertex to v or f_{v_i} belongs to $\{x_1, \ldots, x_n\}$; $\circ \in \{+, -, \times, /\}$ and c is a real constant;

- to every vertex v of outdegree three (*branching vertex*) a test of the form

$$f_w : 0$$

  where w lies on every path from the input vertex to v or $f_w \in \{x_1, \ldots, x_n\}$;

- to every vertex of outdegree zero (*output vertex*) an output.

Edges represent control flow. Given input $\mathbf{a} \in \mathbf{R}^n$, the program will traverse a path $P(\mathbf{a})$ in G from the source to some output vertex. We shall be interested in two ways of assigning a cost to a specific computation. The first supposes that all operations have unit cost, therefore the length of $P(\mathbf{a})$ expresses the cost of the algebraic computation.

More realistically, we may wish to consider the effect of the operands' bit size on the speed of arithmetic operations. We shall say that under the *bit model*, or formally under the *bit model of the algebraic branching program*, there is a cost function on the vertices of the program. Branching, input and output vertices have unit cost. The cost of operation vertices depends on the particular operation executed as well as the bit size of the operands. Let this size be $\mathcal{O}(b)$. Then addition and subtraction have cost $\mathcal{O}(b)$, while the cost of multiplication and division is denoted by $M(b)$. There exists a straightforward algorithm achieving $M(b) = \mathcal{O}(b^{\lg 3})$ and one by Schönhage and Strassen which obtains $M(b) = \mathcal{O}(b \log b \log\log b)$ [12]. The total cost of a computation equals the sum of the costs over all vertices on $P(\mathbf{a})$.

An additional provision is that our model performs exact arithmetic. We regard our perturbation scheme as built on top of an algorithm which works correctly for non-degenerate input and thus can handle round-off errors; see [17] for a relevant discussion.

2.2 Geometric algorithms model

We restrict attention to algorithms in computational geometry and, more precisely, to certain common primitive predicates. Although our results concern these predicates only, we believe that the applicability of our method is significantly wider. In any case, we have

chosen four predicates that cover most geometric problems, including the construction of Convex Hulls, Voronoi Diagrams, Delaunay Triangulations and Hyperplane Arrangements.

The algorithms of interest can be described by an algebraic branching program restricted in two ways. First, the input consists of n real vectors in $\mathbf{R}^d$, each value denoted by $p_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq d$ and n, d positive integers. The input size is then nd. Second, every branching vertex is assigned a polynomial test expression f of exactly one of the following four types.

- Ordering: $f = p_{i,j} - p_{k,j}$.
- Orientation: $f = det\Lambda_{d+1} = det \begin{bmatrix} 1 & p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ 1 & p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & p_{i_{d+1},1} & p_{i_{d+1},2} & \cdots & p_{i_{d+1},d} \end{bmatrix}$.
- Transversality: $f = det\Delta_d = det \begin{bmatrix} p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & & \vdots \\ p_{i_d,1} & p_{i_d,2} & \cdots & p_{i_d,d} \end{bmatrix}$.
- InSphere: $f = det\Gamma_{d+2} = det \begin{bmatrix} 1 & p_{i_1,1} & \cdots & p_{i_1,d} & \sum_{j=1}^{d} p_{i_1,j}^{2j} \\ 1 & p_{i_2,1} & \cdots & p_{i_2,d} & \sum_{j=1}^{d} p_{i_2,j}^{2j} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & p_{i_{d+2},1} & \cdots & p_{i_{d+2},d} & \sum_{j=1}^{d} p_{i_{d+2},j}^{2j} \end{bmatrix}$.

Note that every test expression of these types can be formulated as a determinant in the input variables. The rest of this work is concerned with this restricted version of the algebraic branching program.

2.3 Degeneracy

The notion of degeneracy is hard to formalize because it may depend not only on the particular problem but also on the algorithm chosen to attack it. Geometric notions of degeneracy include the case of more than k hyperplanes having a point in common in k-dimensional euclidean space, which is treated as a special case for Linear Programming too. We then say that these hyperplanes do not intersect transversally and observe that the predicate Transversality is zero.

Definition 2.1 *An input instance is* degenerate with respect to some algorithm *if it causes the test determinant f, which is not identically zero, at some branching vertex of the algorithm's algebraic branching program to evaluate to zero. Equivalently, $\mathbf{a} \in \mathbf{R}^n$ is in* general position *if there is no test determinant f such that $f \not\equiv 0$ and $f(\mathbf{a}) = 0$.*

Let us adopt a more abstract standpoint and consider the space of all input instances. This space is embedded in $\mathbf{R}^{nd}$, equipped with the standard euclidean metric. Each predicate may be regarded as a map of input space to $\{-, 0, +\}$. The preimage of $\{0\}$ under a particular predicate is a lower-dimensional set in $\mathbf{R}^{nd}$, i.e. has codimension at least 1. A point in input space is in general position with respect to this predicate, if there exists an open set containing it, such that the predicate does not change sign on the entire open set. Clearly, every degenerate input for a specific algorithm belongs to the preimage of zero under the mapping of a predicate used by the algorithm.

A problem mapping associates with every input instance a unique output, which is called an (exact) *solution*. The solution may be thought of as the output produced by a solver of infinite computational power. The points at which this mapping fails to be continuous (or does not exist) are exactly the *intrinsic* or *problem-dependent* degeneracies. They form a subset of *algorithm-dependent* degeneracies, as defined above. Thus, this paper concentrates on the latter.

In the example of the planar Convex Hull problem, found also in [17], intrinsic degeneracies occur when 3 or more points are collinear. If a particular algorithm uses a vertical sweep-line or relies on a vertical partition of the plane, then 2 covertical points may constitute an algorithm-dependent degeneracy. A more detailed discussion of the notion of degeneracy appears in [11].

2.4 Problem definition

Given is an algorithm that solves a problem under the assumption of non-degeneracy. Our aim is to directly perturb an arbitrary input instance $\mathbf{a}$ into some other instance $\mathbf{a}'$ in a systematic way so that the same algorithm can always produce a meaningful output. A *valid direct perturbation* must define $\mathbf{a}'$ in general position and satisfy the following condition:

- For non-degenerate inputs, the algorithm produces the same output whether it runs on $\mathbf{a}$ or $\mathbf{a}'$.
- For degenerate inputs, if the problem mapping is continuous, then the algorithm on $\mathbf{a}'$ returns either the exact solution or an output arbitrarily close to it.
- Otherwise, the algorithm produces a correct solution for $\mathbf{a}'$ and the latter is arbitrarily close to $\mathbf{a}$.

In the first case, it suffices to prove that the paths $P(\mathbf{a})$ and $P(\mathbf{a}')$ followed by the algorithm in the algebraic branching program are the same. In the second case, it suffices to show the proximity of $\mathbf{a}$ and $\mathbf{a}'$ in the topology in which the problem mapping is continuous. Note that then the algorithm mapping is also continuous in its own domain. Refer to the cells defined by the preimages of $\{0\}$ under the mappings of predicates employed in the program. Each corresponds to a unique output. The continuity property together with the proximity of the inputs imply the proximity of the respective outputs. In short, we only need to prove for our direct perturbations that each defines non-degenerate instance

$\mathbf{a}'$ arbitrarily close to $\mathbf{a}$, such that when the latter is in general position, $P(\mathbf{a})$ and $P(\mathbf{a}')$ are identical.

To illustrate, consider the Convex Hull problem, for which there is always a solution. Its combinatorial nature causes the output space topology to be discrete. Nonetheless, there exist measures of success in output space, under which the approximate solution is arbitrarily close to the exact one. Take for instance the volume of the symmetric difference between the actual output and the exact convex hull. The perturbed input $\mathbf{a}'$ will be defined in terms of an infinitesimal variable ϵ. Setting ϵ equal to zero makes the symmetric difference volume vanish.

Specific applications may require that the facets of the hull include exactly the points that define them, which implies that extra (degenerate) points should be removed. Then letting ϵ go to zero is not enough; a post-processing phase is necessary in order to produce the solution. Edelsbrunner in [8] discusses this issue. Furthermore, Yap in [17] points out the difficulty encountered with discrete output spaces and indicates the steps that must be formally taken to justify the application of any perturbation method.

2.5 Infinitesimals

Our approach in removing degeneracies is to add to the input values arbitrarily small quantities. To this effect we make use of infinitesimals. The process of extending the field of reals by an infinitesimal is a classic technique, formalized in [2], and used by Canny in [3] and [4].

Definition 2.2 *We call* ϵ infinitesimal *with respect to* $\mathbf{R}$ *if the extension* $\mathbf{R}(\epsilon)$ *is ordered so that* ϵ *is positive but smaller than any positive element of* $\mathbf{R}$.

We write $0 < \epsilon \ll 1$ to indicate that every polynomial in ϵ has value smaller than 1. The sign of a polynomial in ϵ equals the sign of the non-zero term of lowest degree in ϵ. Alternatively, ϵ can belong to the reals and assume an arbitrarily small positive value. Under the algebraic branching program model it is immaterial whether we view ϵ as an infinitesimal or an arbitrarily small real. To see this, consider a special case of Tarski's "Transfer Principle" [16]: There is a finite number of polynomials encountered at branching vertices on $P(\mathbf{a})$, so there is a minimum positive real value among all of their roots. As long as ϵ is smaller than that minimum, it may take any positive value and none of these polynomials will change sign. Hence, the algorithm will follow the same path and, in general, it will behave in the same way as if ϵ was an infinitesimal extension. In this paper, we make use of both standpoints and regard ϵ sometimes as an extension to the reals and sometimes as an arbitrarily small real number.

The general strategy is to transform the real input values into ϵ polynomials. But can we still run the same algorithm on symbolic input? Do we know that it will halt? The answer is affirmative. Due to the above argument, the path traced on the perturbed input will be the same as if we had substituted ϵ by a sufficiently small positive real value. Thus, there exists an actual real input for which the algorithm behaves the same as for the perturbed input.

3 Other approaches

One popular way to handle degeneracies has been to distinguish among the various special cases and take some specific and consistent, though ad-hoc action, every time the objects under examination are not in general position. This is inefficient and ugly. A more common tack is to simply observe that a random perturbation of the input parameters will produce the desired non-degenerate configuration with high probability. This is true for stable problems, in which infinitesimal perturbations do not affect the solution. Nevertheless, this method does not guarantee the removal of all degeneracies.

Symmetry breaking rules in linear programming, such as those of Dantzig in [6], are the earliest systematic approaches to our problem. Although applied only to a particular problem, that method bears a close resemblance to later ones. It perturbs the constant terms by a small amount depending on a symbolic variable ϵ. The perturbation of each constant depends on its unique integer index:

$$b_i(\epsilon) = b_i + \epsilon^i.$$

Edelsbrunner and Mücke systematize in [9] a scheme called Simulation of Simplicity (SoS for short), presented in [7], [10] and [8] which applies to the same class of algorithms as our method. SoS introduces the following perturbation of input parameter $p_{i,j}$:

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

where $\delta \geq d$ and d is the number of dimensions. The sign of the perturbed determinant is the sign of the smallest-degree term in its ϵ-expression. In the worst case, this computation has complexity proportional to $d!$ since there may exist that many terms in the determinant polynomial.

Yap in [17] deals in a more general setting in which the test polynomials are arbitrary and shows, in [18], that his scheme is consistent relative to infinitesimal perturbations. First it imposes a total ordering on all power products

$$w = \prod_{i=1}^{n} v_i^{e_i}, \quad e_i \geq 0$$

where $\mathbf{v} = (v_1, \ldots, v_n)$ is the input. Let $w_1, w_2, \ldots$ be the ordered list of power products larger than 1, i.e. those with at least one positive exponent. Then, each test polynomial $p(\mathbf{v})$ is associated with the infinite list

$$S(p) = (p, p_{w_1}, p_{w_2}, \ldots)$$

where p_{w_k} is the partial derivative of p with respect to w_k. The sign of a non-zero polynomial p is taken to be the sign of the first polynomial in $S(p)$ whose value is not zero, which can always be found after examining a finite number of terms. In the worst case that all partial derivatives have to be computed, the complexity for dense polynomials is exponential in the input size n.

In short, both methods incur a worst-case overhead at least as large as exponential in the input size, under the algebraic branching program model. Our method will be shown to be simpler as well as faster.

4 The perturbation

This section contains the main results. It defines a valid deterministic perturbation to cope with degeneracies in algorithms whose branching decisions depend on the sign of a determinant of one of the four types described above. We introduce a direct perturbation that adds to each input value a small quantity controlled by an infinitesimal variable.

In view of the Ordering predicate, the perturbation quantities must differ for different parameters. One way to achieve this is to make these quantities depend on the indices of the parameters. The Orientation and Transversality predicates are treated together; to remove their degeneracies, when the input consists of n vectors in $\mathbf{R}^d$, it suffices to pick a list of d-vectors, such that every d among them are linearly independent. More specifically, we have to find n such vectors. Then we add one of these vectors to every input vector. Our perturbation scheme below satisfies this requirement and, furthermore, will be shown to remove the degeneracies due to the InSphere predicate.

Let the input instance be grouped into n distinct objects $p_1, p_2, \ldots p_n$ indexed by distinct integers between 1 and n. Every object can be thought of as a point in d-dimensional euclidean space. Every p_i is defined by d real parameters $p_{i,1}, p_{i,2}, \ldots p_{i,d}$, for an arbitrary integer d. Typically n is larger than d. Let c represent the maximum bit size of the input parameters and observe that it is at least as large $\log n$.

Let $p_{i,j}(\epsilon)$ represent the perturbed version of input parameter $p_{i,j}$ where ϵ is a symbolic variable. During computation on the perturbed input ϵ is never evaluated. We define the following perturbation:

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

It is intuitive that an optimal scheme uses as few bits as possible to define the perturbation quantities. These quantities in scheme (1) have bit size $\mathcal{O}(d \log n)$. We can reduce this size to the minimum possible and still obtain n vectors, every d of which are independent. As suggested by Seidel [15], we take the remainder of the above quantities when divided by a large prime. Let this prime p be larger than n. Then the perturbation is defined as

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon(i^j \bmod p). \tag{2}$$

If p is chosen to be close to n, then the perturbation quantities have bit size $\mathcal{O}(\log n)$. This is optimal, since there must be at least nd different quantities.

In what follows we prove the validity of perturbation (1) as well as its computational efficiency with respect to the four predicates of interest. We draw the same conclusions for scheme (2) applied to the Orientation and Transversality predicates.

4.1 Ordering

This predicate decides the order of two parameters expressing the j^{th} coordinate of two input objects. On the perturbed input it has to decide the sign of

$$p_{i,j}(\epsilon) - p_{k,j}(\epsilon) = p_{i,j} + \epsilon i^j - p_{k,j} - \epsilon k^j.$$

The most significant term does not involve ϵ. If this term is zero, though, the predicate compares the factors of the infinitesimal, which comes down to comparing i against k. The perturbed parameters can never be equal, since they are characterized by different indices. Therefore the predicate always returns a non-zero answer by artificially imposing the lexicographic ordering among equal parameters. Moreover, this ordering is consistent. This scheme is not new as applied here. Tie-breaking rules for sorting algorithms are invariably used in practice and lexicographic order is among the most popular systematic ways.

The implementation takes in the worst case, an extra constant-time check, thus doubling the running time of the predicate. Under the bit model, the extra comparison adds a $\mathcal{O}(\log n)$ factor, which again does not affect the predicate's complexity. We have demonstrated the following

Proposition 4.1 *The perturbation defined by (1) satisfies the condition for a valid perturbation with respect to the Ordering predicate and does not change the running-time complexity of this predicate in the algebraic as well as the bit model.*

4.2 Orientation and Transversality

We examine these two predicates together. Orientation decides, given a point and a halfplane in d-dimensional space spanned by d points, in which halfspace defined by the hyperplane the point lies. The proof that the halfspace is uniquely determined by the sign of the determinant, denoted Λ_{d+1}, is explicitly given by Edelsbrunner and Mücke in [9]. Matrix Λ_{d+1} has rows that correspond to input objects $p_{i_1}, p_{i_2}, \ldots p_{i_{d+1}}$:

$$\Lambda_{d+1} = \begin{bmatrix} 1 & p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ 1 & p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & p_{i_{d+1},1} & p_{i_{d+1},2} & \cdots & p_{i_{d+1},d} \end{bmatrix}.$$

The perturbed matrix $\Lambda_{d+1}(\epsilon)$ contains the corresponding perturbed parameters. Its determinant is given by

$$det\Lambda_{d+1}(\epsilon) \;=\; det\Lambda_{d+1} \;+\; (\epsilon^k \; terms, 1 \leq k \leq d-1) \;+\; \epsilon^d det V_{d+1},$$

where V_{d+1} is a $(d+1) \times (d+1)$ Vandermonde matrix with

$$detV_{d+1} = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^d \\ 1 & i_2 & i_2^2 & \cdots & i_2^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_{d+1} & i_{d+1}^2 & \cdots & i_{d+1}^d \end{vmatrix} = \prod_{k>l}(i_k - i_l).$$

Since all indices are distinct positive integers, the Vandermonde matrix is non-singular. Therefore the n perturbation vectors $[i\, i^2 \ldots i^d]$ form a set of vectors, every d of which are

linearly independent. The expression for $det\Lambda_{d+1}(\epsilon)$ is a polynomial in ϵ of degree d, which is not identically zero because of the non-vanishing highest-order term.

There is an alternative way to arrive at the non-singularity of $\Lambda_{d+1}(\epsilon)$ starting from the non-singularity of V_{d+1}. It is shown below by equation (3) that the $det\Lambda_{d+1}(\epsilon)$ can be expressed as a product of $detV_{d+1}$ times a non-vanishing characteristic polynomial in ϵ.

Now consider perturbation (2) applied to the entries of Λ_{d+1}. The perturbed determinant has again a non-vanishing highest-order term in ϵ. It is the determinant of a modified Vandermonde matrix with $[i,k]$-th entry $(i^{k-1} \bmod p)$. Since the determinant is a polynomial in the entries, the determinant of the modified Vandermonde is

$$detV_{d+1} \bmod p \;=\; \prod_{k>l}(i_k - i_l) \bmod p.$$

The prime p, by choice, is larger than any index i_j, thus it exceeds all differences and hence does not divide any. As a consequence, p does not divide their product which implies that the modified Vandermonde and, hence, the Orientation matrix are non-singular. In other words, the set of n vectors obtained from perturbation (2) have the crucial property that every d of them are independent.

Transversality decides whether d hyperplanes in $\mathbf{R}^d$ have a point in common. It is an indispensable predicate for several operations as exemplified in [9]. The $d \times d$ matrix of interest is Δ_d and has rows representing input objects $p_{i_1}, p_{i_2}, \cdots p_{i_d}$:

$$\Delta_d = \begin{bmatrix} p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & & \vdots \\ p_{i_d,1} & p_{i_d,2} & \cdots & p_{i_d,d} \end{bmatrix}.$$

The corresponding matrix $\Delta_d(\epsilon)$ of the perturbed parameters has determinant

$$det\Delta_d(\epsilon) \;=\; det\Delta_d \;+\; (\epsilon^k \; terms, 1 \le k \le d-1) \;+\; \epsilon^d \, detU_d,$$

where U_d is a $d \times d$ matrix, related to the Vandermonde matrix V_d as follows:

$$detU_d = \begin{vmatrix} i_1 & i_1^2 & \cdots & i_1^d \\ i_2 & i_2^2 & \cdots & i_2^d \\ \vdots & \vdots & & \vdots \\ i_{d+1} & i_{d+1}^2 & \cdots & i_{d+1}^d \end{vmatrix} = \prod_{k=1}^{d} i_k \; detV_d$$

Clearly, U_d, as well as V_d, is non-singular since all integer indices are distinct and positive. We have managed again to end up with a matrix whose rows are d of the perturbation vectors. The non-singularity of the matrix is equivalent to the fact that any d such vectors are independent.

The expression for $det\Delta_d(\epsilon)$ is an ϵ-polynomial whose highest order term is non-zero, hence $det\Delta_d(\epsilon)$ is not identically zero. This determinant is given below by (4) as the product of $detV_d$ times a nontrivial characteristic polynomial in ϵ. Since V_d is non-singular, this expression serves as a direct proof of the non-singularity of $\Delta_d(\epsilon)$.

We can also apply perturbation (2) and obtain the same result, since p does not divide the Vandermonde determinant, nor any of the n indices. We are essentially making use of the aforementioned property of the perturbation vectors of (2), namely that every d of them are independent. We are led to

Lemma 4.2 *There exists a positive real constant ϵ_0 such that, for every positive real $\epsilon < \epsilon_0$, every $\Lambda_{d+1}(\epsilon)$ and $\Delta_d(\epsilon)$ matrix occurring at a branching node of the algebraic branching program after perturbation (1) or (2) is non-singular and its determinant has constant sign.*

Proof We shall make repeated use of certain facts from the theory of polynomials over ordered fields. A polynomial which is not identically zero, has a set of roots of codimension 1. Any algebraic set of roots, or zero set, whose dimension is at most 1, is either the entire (real) line or the union of a finite number of points.

The expressions for $det\Lambda_{d+1}(\epsilon)$ and $det\Delta_d(\epsilon)$ are polynomials in ϵ and are not identically zero. Thus, there exists a finite number of roots for each one. Letting ϵ_0 be the minimum positive such root over all test determinants encountered, proves the lemma. □

We now address the question of computing the sign of the perturbed determinant. One obvious way is to evaluate the terms in the determinant's ϵ-expansion in increasing order of the exponent of ϵ. The process stops at the first non-vanishing term and reports its sign. This is the approach adopted by the two other techniques described above, namely SoS [9] and Yap's perturbation [17]. Our perturbation scheme lends itself to a more efficient trick. Namely, we reduce the computation of the perturbed determinant to a characteristic polynomial computation.

$$\begin{aligned} det\Lambda_{d+1}(\epsilon) &= \frac{1}{\epsilon}\begin{vmatrix} \epsilon & p_{i_1,1}(\epsilon) & \cdots & p_{i_1,d}(\epsilon) \\ \vdots & \vdots & & \vdots \\ \epsilon & p_{i_{d+1},1}(\epsilon) & \cdots & p_{i_{d+1},d}(\epsilon) \end{vmatrix} \\ &= \frac{1}{\epsilon} det\left(\begin{bmatrix} 0 & p_{i_1,1} & \cdots & p_{i_1,d} \\ \vdots & \vdots & & \vdots \\ 0 & p_{i_{d+1},1} & \cdots & p_{i_{d+1},d} \end{bmatrix} + \epsilon V_{d+1}\right) \\ &= \frac{1}{\epsilon} det\left(L + \epsilon V_{d+1}\right). \end{aligned}$$

Having implicitly defined L, denoting by I_k the $k \times k$ unit matrix and relying on the fact that every Vandermonde matrix V_{d+1} is invertible, we have

$$\begin{aligned} det\Lambda_{d+1}(\epsilon) &= \frac{1}{\epsilon} det(-V_{d+1})\, det((-V_{d+1}^{-1})L - \epsilon\, I_{d+1}) \\ &= \frac{1}{\epsilon}(-1)^{d+1} det V_{d+1}\, det(M - \epsilon\, I_{d+1}). \end{aligned} \tag{3}$$

Similarly, matrix U_d is used to express

$$det\Delta_d(\epsilon) = det(\Delta_d + \epsilon U_d)$$

$$= det(-U_d)\ det((-U_d^{-1})\Delta_d - \epsilon\, I_d)$$
$$= \prod_{k=1}^{d} i_k\ (-1)^d det V_d\ det(N - \epsilon\, I_d). \qquad (4)$$

Let $MM(k)$ denote the number of operations needed to multiply two $k \times k$ matrices, which is strictly larger than $\mathcal{O}(k^2)$. Recall that $M(b)$ is an upper bound for the number of bit operations required for multiplying two numbers of $\mathcal{O}(b)$ bits each; also that c denotes the bit size of the input values.

Lemma 4.3 *Computing the sign of the determinants of perturbed matrices $\Lambda_{d+1}(\epsilon)$, $\Delta_d(\epsilon)$, obtained from perturbation (1) or (2), can be done in $\mathcal{O}(MM(d))$ steps under the algebraic branching program model. These computations require at most $\mathcal{O}(MM(d)\ (M(c) + M(d^2 \log n))$ bit operations if scheme (1) is used and $\mathcal{O}(MM(d)\ (M(c) + M(d \log n))$ bit operations if (2) is used.*

Proof Calculating $det V_d$ has time complexity $\mathcal{O}(d^2)$ and inverting V_d can be done in $\mathcal{O}(d^2)$ as demonstrated by Zippel in [19]. Inverting $d \times d$ matrix U_d has the same time complexity because the k^{th} column vector of U_d^{-1} is the quotient of the k^{th} column vector of V_d^{-1} divided by i_k. Computing each of M and N takes $\mathcal{O}(MM(d))$ operations. Computing $det(M - \epsilon\, I_{d+1})$ or $det(N - \epsilon\, I_d)$ is a characteristic polynomial computation for which there exists an algorithm by Keller-Gehrig [14] which requires $\mathcal{O}(MM(d))$ operations. Putting everything together requires at most $\mathcal{O}(d)$ time, since the result is a polynomial in ϵ of $\mathcal{O}(d)$ terms.

Now consider the sizes in bits of the numbers involved and concentrate on the overhead incurred by the perturbation quantities only in case scheme (1) is employed. The introduced quantities have bit size $\mathcal{O}(d \log n)$ and the entries of the inverted Vandermonde matrix have bit size $\mathcal{O}(d^2 \log n)$. Therefore, the computations of $det V_d$ and $-V_d^{-1}$ cost $\mathcal{O}(d^2 M(d^2 \log n))$ under the bit model. Computing M, N or the respective characteristic polynomials takes $\mathcal{O}(MM(d))$ times $\mathcal{O}(M(d^2 \log n) + M(c))$ bit operations. In calculating the final polynomial the bottleneck is the execution of $\mathcal{O}(d)$ multiplications, with $\mathcal{O}(d^2 \log n + c)$-bit numbers at worst, which has complexity $\mathcal{O}(d)\mathcal{O}(M(d^2 \log n) + M(c))$. Clearly, the most expensive part of the entire computation under both models is the matrix multiplication to compute M or N and the characteristic polynomial calculation.

When scheme (2) is used, the original bit size of the perturbation quantities is $\mathcal{O}(\log n)$ and the size of entries in the inverted Vandermonde matrix is $\mathcal{O}(d \log n)$. The bottleneck is again the matrix multiplication to compute M and N, and the total complexity is $\mathcal{O}(MM(d))\mathcal{O}(M(c) + M(d \log n))$. □

Proposition 4.4 *The perturbations defined by (1) and (2) satisfy the condition for a valid perturbation and do not change the running-time complexity of the Orientation and Transversality predicates under the algebraic branching program model. Under the bit model and scheme (1), the bit complexity increases by factor $\mathcal{O}(d^{2+\alpha})$ for any positive real constant α. With scheme (2), the overhead factor is $\mathcal{O}(d^{1+\alpha})$.*

Proof At branching nodes, the sign of the perturbed determinant is taken to be the sign of the lowest order non-vanishing term. If the original determinant is non-zero, it dominates the ϵ-polynomial. Otherwise, we simulate an artificial non-degenerate situation in which the input points are arbitrarily close to the original ones. Lemma 4.2 proves the consistency of our answers as ϵ approaches to zero on the positive real axis.

Lemma 4.3 asserts that the implementation takes $\mathcal{O}(MM(d))$ operations which is the original algebraic complexity of the algorithm since it had to compute a $d \times d$ determinant.

With respect to the bit complexity, recall that c, which is at least as large as $\log n$, is the bit size of the input parameters. The original bit complexity per operation is thus $M(\log n)$. On perturbed input the worst-case complexity is $M(d^2 \log n)$ or $M(d \log n)$, depending on whether scheme (1) or (2) is employed. Now, $M(d^2 \log n)$ is $\mathcal{O}(d^2 \log n\ (\log d + \log\log n)\ (\log d + \log\log\log n)) = \mathcal{O}(d^2 \log n\ \log^2 d\ \log\log n\ \log\log\log n) = \mathcal{O}(d^2 \log^2 d\ M(\log n))$. A similar argument shows that the factor induced by scheme (2) is bounded by $\mathcal{O}(d \log^2 d) M(\log n)$. Hence the extra factor due to the perturbation is $\mathcal{O}(d^{2+\alpha})$ or $\mathcal{O}(d^{1+\alpha})$ respectively, for an arbitrarily small positive constant α. □

Corollary 4.5 *The bit complexity of predicates Orientation and Transversality is unaffected by perturbations (1) and (2) if d is $\mathcal{O}((c/\log n)^{1/2})$ or $\mathcal{O}(c/\log n)$ respectively.*

Proof Immediate from lemma 4.3, the fact that the original complexity is $\mathcal{O}(MM(d)\ M(c))$ and the lower bound $\log n$ on c. □

4.3 InSphere

This predicate decides whether the $(d+2)$nd point lies in the interior of the higher-dimensional sphere of the first $d+1$ points. Clearly, $d+1$ points define a sphere in $\mathbf{R}^d$. We can lift all points in $d+1$ dimensions, on the surface of a paraboloid, by adding a $(d+1)$st coordinate equal to the sum of the squares of the d coordinates that describe the input point. Then, testing whether a point lies in the interior of a sphere reduces to checking, in $d+1$ dimensions, whether it lies below the hyperplane of the $d+1$ points. Hence we obtain the formulation of the predicate as a determinant. A more detailed proof is found in [9].

The first $d+1$ rows correspond to the points defining the sphere, after being lifted to $(d+1)$-dimensional space. The last row has the coordinates of the lifted query point. The Γ_{d+2} matrix is $(d+2) \times (d+2)$:

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

The determinant of the perturbed matrix $\Gamma_{d+2}(\epsilon)$ is a polynomial in ϵ.

$$det\Gamma_{d+2}(\epsilon) = det\Gamma_{d+2} + (\epsilon^k\ terms, 1 \le k \le d+1) + \epsilon^{d+2} detW_{d+2},$$

where W_{d+2} resembles a Vandermonde matrix and is actually

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

To prove that $det\Gamma_{d+2}(\epsilon)$ is not identically zero, it suffices to show that W_{d+2} is non-singular. Assume that it is. Then there exists a non-zero $(d+2)$-vector $\mathbf{z}$ in its kernel. Fix $\mathbf{z} = [z_0\ z_1\ \dots\ z_{d+1}]$. Then the following univariate polynomial in x,

$$z_0 + xz_1 + x^2z_2 + \cdots + x^dz_d + \sum_{j=1}^{d} x^{2j}z_{d+1},$$

has at least $d+2$ distinct positive integer solutions, namely $i_1, \dots, i_{d+2}$.

To derive a contradiction consider Descartes' rule of sign, as presented by Collins and Loos in [5]. It states that the number of sign variations of a univariate polynomial's coefficients exceeds the number of positive zeros, multiplicities counted, by an even non-negative integer. To define the number of sign variations of a (finite) sequence of real numbers $< u_1, \dots, u_s >$, let $< u'_1, \dots, u'_t >$ be the subsequence of all non-zero elements. Then the number of variations is the number of consecutive element pairs u_k, u_{k+1}, for $1 \leq k < t$, such that the product $u_k u_{k+1}$ is negative.

Applying Descartes' rule we conclude that the number of sign variations must be at least equal to $d+2$, hence there must exist at least $d+3$ non-zero distinct coefficients. We rewrite the polynomial in x in its standard form:

$$z_0 + xz_1 + x^2(z_2 + z_{d+1}) + x^3z_3 + x^4(z_4 + z_{d+1}) + \cdots + \sum_{j=\lceil d/2 \rceil}^{d} x^{2j}z_{d+1}.$$

Obviously, there are at most $d+2$ distinct coefficients, which leads to a contradiction. Therefore the hypothesis about the existence of vector $\mathbf{z}$ is false, which implies that the kernel of the linear transformation is the zero set. Equivalently, the matrix of the transformation, W_{d+2}, is non-singular, which implies that $det\Gamma_{d+2}(\epsilon)$ is not identically zero.

There is an alternative proof to the claim that $\Gamma_{d+2}(\epsilon)$ is not identically zero. Consider expression (5) below for $det\Gamma_{d+2}(\epsilon)$ as the sum of two determinants. The first is $det\Gamma_1$ expressed in (6) as the product of $detW_{d+2}$ times a characteristic polynomial in ϵ. This is not identically zero, since W_{d+2} was just shown to be non-singular and the polynomial has at least one non-vanishing term. A similar argument stands for $det\Gamma_2$, based on formula (7). Hence, we have established the non-singularity of $\Gamma_{d+2}(\epsilon)$ in two ways.

Although W_{d+2} behaves similarly enough to a Vandermonde so as to be non-singular, its determinant is not as well-behaved. There is no clean expression for it and it does not seem that we can extend scheme (2) in order to cover the InSphere predicate. Thus, we restrict attention to scheme (1) only.

Lemma 4.6 *There exists a positive real constant ϵ_0 such that, for every positive real $\epsilon < \epsilon_0$, the determinant of a $\Gamma_{d+2}(\epsilon)$ matrix, obtained from perturbation (1) is non-zero and has constant sign.*

Proof We showed above that the perturbed determinant does not vanish identically because its expression as an ϵ polynomial always contains a non-zero term, namely the highest-degree term. The existence of ϵ_0 follows from an argument identical to that for lemma 4.2. □

We still have to tackle the question of implementing the InSphere predicate for perturbed input. We shall try again to reduce the computation to that of a characteristic polynomial.

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

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

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

$$= det\Gamma_1 + det\Gamma_2. \tag{5}$$

Taking advantage of the linearity of the determinant in the last column, we have reduced the computation to two parts. Each is reduced to a characteristic polynomial computation.

$$\begin{aligned} det\Gamma_1 &= \begin{vmatrix} \epsilon & p_{i_1,1}(\epsilon) & \dots & p_{i_1,d}(\epsilon) & \epsilon \sum_{j=1}^{d} i_1^{2j} + (2\sum_{j=1}^{d} p_{i_1,j} i_1^j) - i_1^{d+2} \\ \vdots & \vdots & & \vdots & \vdots \\ \epsilon & p_{i_{d+2},1}(\epsilon) & \dots & p_{i_{d+2},d}(\epsilon) & \epsilon \sum_{j=1}^{d} i_{d+2}^{2j} + (2\sum_{j=1}^{d} p_{i_{d+2},j} i_{d+2}^j) - i_{d+2}^{d+2} \end{vmatrix} \\ &= det\left(\begin{bmatrix} 0 & p_{i_1,1} & \dots & p_{i_1,d} & (2\sum_{j=1}^{d} p_{i_1,j} i_1^j) - i_1^{d+2} \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & p_{i_{d+2},1} & \dots & p_{i_{d+2},d} & (2\sum_{j=1}^{d} p_{i_{d+2},j} i_{d+2}^j) - i_{d+2}^{d+2} \end{bmatrix} \right. \\ &\quad + \left. \epsilon \begin{bmatrix} 1 & i_1 & \dots & i_1^d & \sum_{j=1}^{d} i_1^{2j} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & i_{d+2} & \dots & i_{d+2}^d & \sum_{j=1}^{d} i_{d+2}^{2j} \end{bmatrix} \right) \\ &= det(G_1 + \epsilon W_{d+2}) \\ &= det(-W_{d+2})\ det((-W_{d+2}^{-1})G_1 - \epsilon I_{d+2}). \end{aligned} \tag{6}$$

Note that the inversion of W_{d+2} is justified by its non-singularity that was demonstrated previously. The computation of $det\Gamma_2$ is very similar to that of $det\Gamma_1$ and is omitted. The resulting expression is

$$det\Gamma_2 = \frac{1}{\epsilon}\, det(-V_{d+2})\; det((-V_{d+2}^{-1})G_2 - \epsilon I_{d+2}), \tag{7}$$

where V_{d+2} is the $(d+2) \times (d+2)$ Vandermonde matrix and

$$G_2 = \begin{bmatrix} 0 & p_{i_1,1} & \cdots & p_{i_1,d} & \sum_{j=1}^{d} p_{i_1,j}^2 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & p_{i_{d+2},1} & \cdots & p_{i_{d+2},d} & \sum_{j=1}^{d} p_{i_{d+2},j}^2 \end{bmatrix}.$$

Recall that $MM(k)$ denotes the complexity of multiplying two $k \times k$ matrices under the algebraic model, while in the bit model the multiplication of two b-bit numbers costs $M(b)$.

Lemma 4.7 *Computing the sign of the determinant of perturbed matrix $\Gamma_{d+2}(\epsilon)$, obtained from perturbation (1), under the algebraic branching program model takes $\mathcal{O}(MM(d))$ time. The complexity under the bit model is $\mathcal{O}(MM(d)\ (M(c) + M(d^2 \log n))$.*

Proof Repeating the arguments of lemma 4.3 we conclude that the bottleneck occurs in three points where we have to perform an operation as costly as matrix multiplication. First, we have to invert W_{d+2}, which can be done simply in $\mathcal{O}(MM(d))$, without exploiting its near-Vandermonde structure. Second, we have to compute the matrix products $(-W_{d+2}^{-1})\, G_1$ and $(-V_{d+2}^{-1})\, G_2$. Finally, the characteristic polynomial algorithm is executed twice. The overall asymptotic cost is therefore $\mathcal{O}(MM(d))$, although there is an extra factor of 2, which we did not have with the Orientation and Transversality predicates.

The bit size of all quantities involved here, including the sum of squares, is at most $\mathcal{O}(\log dn^{2d}) = \mathcal{O}(d \log n)$. Hence the bit complexity of the InSphere predicate is the same as that for computing Orientation and Transversality under perturbation (1). □

Two implementation remarks are in order here. To exploit the structure of W_{d+2} we may regard it as the product of two square matrices W_1 and W_2:

$$W_{d+2} = \begin{bmatrix} V_{d+1} & B \\ C^T & D \end{bmatrix} = \begin{bmatrix} V_{d+1} & 0 \\ C^T & 1 \end{bmatrix} \begin{bmatrix} I_{d+1} & V_{d+1}^{-1}B \\ 0 & D - C^T V_{d+1}^{-1} B \end{bmatrix} = W_1 W_2.$$

Recall that V_{d+1} stands for a $(d+1) \times (d+1)$ Vandermonde matrix; B and C denote $(d+1) \times 1$ vectors and D is a single entry. Let $A = D - C^T V_{d+1}^{-1} B$. Then it is easier to compute the inverse, although at the same complexity cost.

$$W_{d+2}^{-1} = W_2^{-1} W_1^{-1} = \begin{bmatrix} I_{d+1} & -V_{d+1}^{-1} B A^{-1} \\ 0 & A^{-1} \end{bmatrix} \begin{bmatrix} V_{d+1}^{-1} & 0 \\ -C^T V_{d+1}^{-1} & 1 \end{bmatrix}.$$

In case that we choose not to use Keller-Gehrig's algorithm [14] for the characteristic polynomial but instead apply the simpler $\mathcal{O}(d^3)$ method, then the two computations may be combined, because the involved matrices have several similarities. It is not clear how to combine the two applications of Keller-Gehrig's method. Anyhow, this would achieve only a constant factor improvement.

Proposition 4.8 *The perturbation defined by (1) is valid and does not affect the complexity of the InSphere predicate under the algebraic branching program model. Under the bit model, the complexity increases by factor $\mathcal{O}(d^{2+\alpha})$, for an arbitrarily small positive constant α.*

Proof The perturbation scheme does not affect the outcome of InSphere when the input is in general position. Otherwise, it simulates an artificial configuration for which the InSphere matrix is non-singular for sufficiently small ϵ, as asserted by Lemma 4.6. As ϵ grows arbitrarily small, this configuration tends to be identical with the unperturbed one.

Lemma 4.7 guarantees that the complexity in the algebraic model is unaffected since InSphere must spend $\mathcal{O}(MM(d))$ time to compute the unperturbed determinant. The original bit complexity per operation is $\mathcal{O}(M(c))$ and c is at least as large as $\log n$. On perturbed input the worst-case complexity is $M(d^2 \log n)$, which is bounded by $\mathcal{O}(d^2 \log^2 d)M(\log n)$. Hence the overhead factor is $\mathcal{O}(d^{2+\alpha})$ for any arbitrarily small positive constant α. □

Corollary 4.9 *If $d = \mathcal{O}((c/\log n)^{1/2})$, the bit complexity is unaffected.*

Proof Immediate from lemma 4.7 and the proof of proposition 4.8. □

We now summarize the main results of the section. Perturbation (1) is valid as applied to all four predicates of interest and incurs no extra running-time complexity under the algebraic branching program model. Under the bit model, when d is significantly larger than $(c/\log n)^{1/2}$, then the complexity becomes $\mathcal{O}(MM(d)M(c)d^{2+\alpha})$, where the original complexity was $\mathcal{O}(MM(d)M(c))$ and α is an arbitrarily small positive constant.

Perturbation (2) is applied to Orientation and Transversality and its validity proven. Again, it does not affect the algebraic complexity but may incur an overhead factor under the bit model, asymptotically as large as $\mathcal{O}(d^{1+\alpha})$, in case that d significantly exceeds $c/\log n$.

5 Applications

We consider the application of our perturbation to some specific problems in 2-dimensional space and examine the issue of post-processing. This issue is also discussed by Edelsbrunner in [8] and Yap in [17]. It is clear that, having perturbed the input by introducing an artificial symbolic variable, we obtain an approximate solution. For certain applications though, we may wish to obtain the exact solution by going through a post-processing stage.

Certain applications do not require such a phase, however. An example is the problem of Delaunay Triangulation on the plane, that can be solved by algorithms using some of the four predicates studied here, such as Guibas and Stolfi's method [13]. A common degeneracy is to have more than 3 cocircular points. They form a convex polygon that must be arbitrarily triangulated. Applying scheme (1) we avoid this special case and output a full triangulation without need of post-processing.

Constructing a line arrangement uses Ordering, Orientation and Transversality, therefore scheme (2) can be of use. The perturbation has the side-effect of creating vacuous

faces, namely planar cells of surface area proportional to ϵ, thus approaching zero. These faces are removed during the post-processing phase.

Lastly, we return to our running example of Convex Hulls, and focus on the planar case. The main predicate is Orientation, hence scheme (2) may be applied. In the Introduction we discerned two types of side-effects. If points are perturbed away from lines so that they lie in the interior of the hull, their distance from the closest edge depends on ϵ. Post-processing moves these points back on the lines by letting the symbolic variable go to zero. The second scenario is that certain edges form vanishingly small angles, and these are collapsed into a single edge. Post-processing takes linear time provided that we have gone under adequate book-keeping during the execution of the algorithm, recording the relations among the unperturbed input points.

6 Conclusion

We have studied the question of degeneracy in relation with algorithms in computational geometry. More precisely, we focused on algorithms modeled as algebraic branching programs with inputs from an infinite ordered field and test expressions that can be formulated as one of four specific determinants. These four predicates cover most important geometric problems.

The main contribution of this work is a direct deterministic perturbation scheme discerned for its conceptual simplicity and computational efficiency. It improves upon existing methods in both respects. The main idea is to alter the input objects by adding vanishingly small quantities. Then we prove the scheme's validity for the four tests of interest and show that it does not affect their algebraic complexity. Under the bit model the implementation of the perturbation may result in an extra overhead, which depends on d. Even if d is not constant, it is usually a small integer, independent of n; then the complexity is asymptotically the same. A variation of this scheme allowed us to optimize the bit size of the perturbation quantities. We proved the applicability of the new scheme to the Orientation and Transversality predicates. Its computational overhead is even smaller.

There are a few directions for future work on the problem of coping with degeneracies. First, we may try to obtain a perturbation for the IsSphere predicate that optimizes the bit size of the perturbation quantities. With Canny, we have managed to generalize our approach over arbitrary polynomial tests in [11]. Canny's work on semi-algebraic sets [4] covers another aspect of the problem. It is interesting to attempt extending the notion of degeneracy over finite fields, in which the lack of order makes our definition of degeneracy invalid. Another direction of generalization is to observe that the output vertices in our model are associated with semi-algebraic sets defined by the test polynomials encountered on the path from the input vertex; we may wish to perturb these sets into general position.

Acknowledgment

I wish to thank my advisor, Professor John Canny, for the invaluable guidance during the present research project. I also thank the readers of the master's thesis, Professor Raimund

Seidel for many useful comments as well as for suggesting to look at the InSphere predicate, and Professor Brian Barsky.

References

[1] Blum L., M. Shub and S. Smale, On a Theory of Computation and Complexity over the Real Numbers: NP Completeness, Recursive Functions and Universal Machines, *Bull. AMS*, Vol. 21, No. 1, pp. 1-46, July 1989.

[2] Bochnak J., M. Coste and M.F. Roy, *Géométrie algébrique réelle*, No. 12, Ergebnisse der Mathematik 3, Springer-Verlag, Berlin, 1987.

[3] Canny J., Some Algebraic and Geometric Computations in PSPACE, *Proc. 20th Annual ACM STOC*, pp. 460-467, 1988.

[4] Canny J., Computing Roadmaps of Semi-Algebraic Sets, to appear in *Proc. 9th AAECC*, New Orleans, Oct. 1991.

[5] Collins G.E. and R. Loos, Real Zeros of Polynomials, *Computer Algebra: Symbolic and Algebraic Computation*, ed. B. Buchberger, G.E. Collins, R. Loos and R. Albrecht, Springer-Verlag, Wien, pp. 83-94, 1982.

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

[7] Edelsbrunner H., Edge-skeletons in arrangements with applications, *Algorithmica 1*, pp. 93-109, 1986.

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

[9] Edelsbrunner H. and E.P. Mücke, Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, *ACM Trans. Graphics*, Vol. 9, No. 1, pp. 67-104, 1990.

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

[11] Emiris I. and J. Canny, A General Approach to Removing Degeneracies, to appear in *Proc. 32nd Annual IEEE FOCS*, San Juan, Oct. 1991.

[12] Gårding L. and T. Tambour, *Algebra for Computer Science*, Springer-Verlag, New York, 1988.

[13] Guibas L. and J. Stolfi, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, *ACM Trans. Graphics*, Vol. 4, No. 2, pp. 74-123, 1985.

[14] Keller-Gehrig W., Fast Algorithms for the Characteristic Polynomial, *Theor. Comp. Sci.*, Vol. 36, pp. 309-317, 1985.

[15] Seidel R., private communication, 1991.

[16] Tarski A., *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, 1948.

[17] Yap C.-K., Symbolic treatment of geometric degeneracies, *Proc. 13th IFIP Conf. on Sys. Modeling and Optimization*, Tokyo, 1987.

[18] Yap C.-K., A geometric consistency theorem for a symbolic perturbation scheme, *Proc. 4th ACM Symp. on Comp. Geometry*, Urbana, 1988.

[19] Zippel R., Interpolating Polynomials from their Values, *J. Symbolic Computation*, No. 9, pp. 375-403, 1990.