In *Proceedings of the 5th International Conference in Central Europe on Computer Graphics and Visualization, WSCG'97*, pages 386–395, Plzeň, Czech Republic, February 1997.

Symbolic Perturbation and Special Arithmetics for Controlled Handling of Geometric Degeneracies

Michael A. Neuhauser†

Vienna University of Technology, Inst. of Computer Graphics, Algorithms and Programming Methodology Group, Karlsplatz 13/1861, A–1040 Vienna, Austria
email: mike@gams.at

Abstract

Descriptions of geometric algorithms in computational geometry texts often exclude the detailed handling of degenerate cases. With symbolic perturbation it suffices to consider only general cases. In this paper we present a symbolic perturbation scheme that is valid for all algorithms with rational functions as branching conditions. We conceptually introduce an already implemented perturbation library that handles all degeneracies automatically and transparently to the programmer. The library ensures correct and efficient computations by the combination of interval and exact rational arithmetic. The influence of perturbation on the solution of geometric problems is investigated; we infer that it is not possible to reconstruct the correct solution for some problems when perturbation is applied to them. A novel technique called problem-specific perturbation is presented to overcome this deficiency; it allows a controlled handling of degeneracies. The perturbation library can be adapted easily to support problem-specific perturbation. The method is applied to the problem of finding all pairs of intersecting line segments, so that the correct result is obtained while no degeneracies have to be considered.

1 Introduction

Implementing computational geometry algorithms based on the description in textbooks or papers often requires considerable effort. Mostly this is the case because the theoretical aspects of algorithms (i.e. the global structure of the problem and complexity analysis) are investigated primarily. To ease this kind of investigations it is usually assumed that computations can be performed exactly (i.e. with arbitrary precision), and that the geometrical objects which are the input data of the algorithm are in a general position (e.g., no three points lie on the same line).

In real world computing none of these assumptions are fulfilled. Floating point arithmetic with limited precision is the standard arithmetic provided by today's programming languages. Input data that comes from man-made structures (CAD, VLSI design, etc.) is highly regular and therefore contains many degenerate (or special) cases. Thus, it is important that the implementation of an algorithm works for all possible inputs.

Symbolic perturbation is a recent technique to handle degenerate cases [3, 9, 4] (an overview is given in [7]). This technique demands an exact arithmetic, but the cited papers do not go into too much detail about this. One promising approach to exact *and* efficient arithmetic is the combination of interval floating point arithmetic and rational arithmetic with arbitrary precision (called lazy arithmetic [1]). We will show how these two techniques can be combined to serve for the implementation of geometric algorithms.

†Supported by project P10782–MAT of the "Fonds zur Förderung der wiss. Forschung (FWF)".

The outline of the paper is as follows. Firstly, we will define a model of computation for geometric algorithms and give an introduction to symbolic perturbation. Then we will present a symbolic perturbation scheme that handles all degenerate cases automatically for rational functions by means of a C++ class library. Next we will analyse the influence of the perturbation on the result of geometric algorithms. It will be shown that generic perturbation can not be applied straightforwardly to some geometric problems. A new approach to overcome this deficiency called *problem-specific perturbation* will be presented next. This method will be applied to line segment intersection.

2 Model of computation and topological primitives

It is necessary to define a model of computation for geometric algorithms so we can deal with them in general. We will use a model called *extended algebraic decision trees* [6] that is able to represent most algorithms. An algorithm is represented as a finite ternary tree in this model. Computation starts at the root of the tree. At each interior node the sign of a real valued function is computed from the input data (e.g. coordinates of points or other geometric objects). This sign determines which of the three outgoing branches (that are labelled with $+1$, 0, and -1, respectively) is selected to get to the next node. When a leaf node is reached the result is computed from the input data by a result function specific to this leaf. The result can be a geometrical structure (e.g. a convex hull), an answer to an intersection query, etc.

The functions that are computed in the interior nodes are called *topological primitives* [3], because they provide topological information about the input data. The primitives are defined and continuous for the encountered values; typically polynomials or rational functions are used.

An example for a topological primitive is *left-of*; a function that has three points as input data and computes the signed distance of the first point from the directed line going from the second to the third point. The sign of this primitive tells whether the first point is left or right of the line; this is the general case. If the sign is zero the point is *on* the line; this is the degenerate case.

A sign of zero always announces a degenerate case for any topological primitive, because these cases are the only discontinuities of the sign function (the topological primitive itself is continuous).

The input data of the algorithm is denoted by $\bar{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, $n \geqslant 1$. The modelling tree of any algorithm and the size of any input data is finite. Hence, there is always a finite number $k \geqslant 1$ of topological primitives $t_i(\bar{x})$. We use a fixed but arbitrary algorithm throughout the paper. Thus, the values of n and k are fixed, as well as the topological primitives t_i, $1 \leqslant i \leqslant k$.

3 Symbolic perturbation

How can symbolic perturbation help in the implementation of geometric algorithms? Suppose you would like to implement an algorithm from a general description given in a textbook or paper, where the handling of degenerate cases is not specified. The extended algebraic decision tree corresponding to the described algorithm will have no branches labelled with 0. So the algorithm's behaviour is undefined if a degenerate case causes a topological primitive to return zero. Severe topological inconsistencies can arise if one of the two other branches ($+1$ or -1) is chosen arbitrarily.

Symbolic perturbation guarantees that each call to a primitive will return either a positive or a negative value, and that all these values are topologically consistent. This is achieved by perturbing the original input data, i.e. every value of the data is changed a little bit by adding a symbolic (not numeric!) value. *Every* computation in the algorithm is then performed with the perturbed data. This ensures the topological consistency.

An example should clarify this. Suppose we need to find out which is the larger of two coordinates x_1 and x_2. We use a primitive that computes $x_1 - x_2$; its sign agrees with the three possible cases: $+1$ if $x_1 > x_2$, -1 if $x_2 > x_1$, and 0 for the degenerate case $x_1 = x_2$. The perturbation substitutes $x_1 + \varepsilon$ for x_1 and $x_2 + \varepsilon^2$ for x_2, where ε is an infinitesimally small positive symbolical entity, i.e. no numerical value is ever assigned to it. The primitive now computes $x_1 + \varepsilon - x_2 - \varepsilon^2$. Only the sign of this expression is needed in the algorithm, and it can be determined without knowing the numerical value of ε. If $x_1 \neq x_2$ then $|x_1 - x_2| > \varepsilon - \varepsilon^2$, because ε is infinitesimally small. Thus, $\operatorname{sgn}(x_1 + \varepsilon - x_2 - \varepsilon^2) = \operatorname{sgn}(x_1 - x_2)$, i.e. the sign is the same as for the unperturbed data. If $x_1 = x_2$ then $\operatorname{sgn}(x_1 + \varepsilon - x_2 - \varepsilon^2) = \operatorname{sgn}(\varepsilon - \varepsilon^2) = +1$. Hence, the sign of the perturbed primitive is never 0.

In general, perturbation substitutes coordinates with functions in ε.

Definition 1 A *perturbation scheme* assigns to every $x \in \mathbb{R}$ a continuous function $x(\varepsilon)$ with $x(0) = x$. The function $x(\varepsilon)$ is called the *perturbation* of x. For every $\bar{x} \in \mathbb{R}^n$ we define $\bar{x}(\varepsilon) := (x_1(\varepsilon), x_2(\varepsilon), \ldots, x_n(\varepsilon))$.

Whenever the algorithm computes $\operatorname{sgn} t_i(\bar{x})$ as a branching condition, the perturbed algorithm has to compute

$$\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)).$$

Note that no mixed use of $\bar{x}$ and $\bar{x}(\varepsilon)$ is allowed; this ensures topological consistency.

The purpose of perturbation is the elimination of all degenerate cases. To achieve this, the input data is changed. Hence, the result for the original and the perturbed data will not necessarily be the same. To minimize the difference, the perturbation should not change the signs of primitives in non-degenerate cases.

Definition 2 A perturbation scheme is called *valid* iff for every $\bar{x} \in \mathbb{R}^n$ and every topological primitive t_i of an algorithm the following two properties are satisfied:

- No degenerate cases are encountered for the perturbed data:

$$\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) \neq 0.$$

- The signs of topological primitives are not changed in non-degenerate cases:

$$t_i(\bar{x}) \neq 0 \implies \lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \operatorname{sgn} t_i(\bar{x}).$$

4 A perturbation scheme for rational functions

Now we will present a perturbation scheme that is valid for any algorithm where the topological primitives are all rational functions. We will show how the value of perturbed primitives (i.e. $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon))$) can be computed, therefore enabling the practical application of this perturbation.

The results we present in this section are closely related to those of Yap [9, 8], but the mathematics used here are much less complicated, while nothing of practical value is lost. Also, topological consistency follows more easily. Yap's scheme for rational functions [8] has a real deficiency. It requires that a rational function is defined in the unperturbed case for perturbation to be applicable. This restricts the usefulness of Yap's perturbation method, as one can see from the following example. Given two lines, each specified by two points, the intersecting point of the lines is to be computed. The expressions for the coordinates of this point are rational functions, whose denominators are equal to zero if the lines are parallel. After perturbation the lines will not be parallel anymore, i.e. they will intersect. But Yap's scheme does not allow to use the intersection point, because the rational functions for the coordinates of the unperturbed intersecting point are undefined. We overcome this deficiency with the following theorem.

Theorem 1 *If a perturbation scheme is valid for polynomials as primitives, the same perturbation scheme is also valid for rational functions as primitives, and for* $t_i(\bar{x}) = \frac{p_i(\bar{x})}{q_i(\bar{x})}$

$$\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \frac{\lim_{\varepsilon \to 0^+} \operatorname{sgn} p_i(\bar{x}(\varepsilon))}{\lim_{\varepsilon \to 0^+} \operatorname{sgn} q_i(\bar{x}(\varepsilon))} \tag{1}$$

holds for $1 \leqslant i \leqslant k$ *and every* $\bar{x} \in \mathbb{R}^n$*. Hence, the sign of a perturbed rational function can be computed from the signs of the perturbed numerator and denominator polynomials.*

Proof: Let $i \in \{1, 2, \dots k\}$ and $x \in \mathbb{R}^n$ be fixed but arbitrary. As the perturbation scheme is valid for polynomials, $\lim_{\varepsilon \to 0^+} \operatorname{sgn} q_i(\bar{x}(\varepsilon)) \neq 0$. Hence, (1) holds. The validity for rational primitives follows right from this and the validity for polynomials ($\lim_{\varepsilon \to 0^+} \operatorname{sgn} p_i(\bar{x}(\varepsilon)) \neq 0$). $\square$

From now on we will assume that all t_i are polynomials, and show the validity of the specific perturbation scheme $\bar{x}(\varepsilon) := \bar{x} + \varepsilon^{\bar{c}}$, with $\bar{c} \in \mathbb{N}_+^n$ and $\varepsilon^{\bar{c}} := (\varepsilon^{c_1}, \varepsilon^{c_2}, \dots, \varepsilon^{c_n})$. Because of Theorem 1 the same perturbation scheme is valid if an algorithm has rational functions as primitives. They just have to be split up in numerator and denominator polynomials.

The value of $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon))$ will be determined by computing some partial derivatives of t_i at $\bar{x}$ in a specific order. We will use a very compressed notation to simplify the formulas. Let G be the largest degree of any primitive and any variable. With $\mathbb{E} := \{0, 1, \dots, G\}^n$ and $|\bar{e}| := \sum_{j=1}^n e_j$, $\bar{e} \in \mathbb{E}$, any non-zero partial derivative of any primitive can written as

$$\mathrm{D}^{\bar{e}} t_i(\bar{x}) := \frac{\partial^{|\bar{e}|} t_i(\bar{x})}{\partial x_1^{e_1} \cdots \partial x_n^{e_n}} \qquad \bar{e} \in \mathbb{E}.$$

Definition 3 A total ordering $\prec$ on $\mathbb{E}$ is called an *admissible ordering* iff there is a $\bar{c} \in \mathbb{N}_+^n$ that *induces* $\prec$, i.e. (with $\langle \bar{c}, \bar{e} \rangle := \sum_{j=1}^n c_j e_j$)

$$\bar{e} \prec \bar{f} \iff \langle \bar{c}, \bar{e} \rangle < \langle \bar{c}, \bar{f} \rangle \qquad \forall\, \bar{e}, \bar{f} \in \mathbb{E}.$$

Theorem 2 *For every admissible ordering* $\prec$ *there exists a* $\bar{c} \in \mathbb{N}_+^n$ *such that:*

1) $\bar{x}(\varepsilon) = \bar{x} + \varepsilon^{\bar{c}}$ *is a valid perturbation scheme.*

2) *For every primitive* t_i *and every* $\bar{x} \in \mathbb{R}^n$ *there is a* $\bar{s} \in \mathbb{E}$ *with* $\mathrm{D}^{\bar{s}} t_i(\bar{x}) \neq 0$ *and* $\mathrm{D}^{\bar{e}} t_i(\bar{x}) = 0$ *for all* $\bar{e} \in \mathbb{E}$, $\bar{e} \prec \bar{s}$, *such that*

$$\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x}).$$

Proof: Let $\bar{c} \in \mathbb{N}_+^n$ induce $\prec$ according to Definition 3. Let $i \in \{1, 2, \dots k\}$ and $x \in \mathbb{R}^n$ be fixed but arbitrary.

Firstly we show 2). Applying the Taylor formula to $t_i(\bar{x}(\varepsilon))$ yields

$$t_i(\bar{x} + \varepsilon^{\bar{c}}) = \sum_{\bar{e} \in \mathbb{E}} \tfrac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x}) \left(\varepsilon^{\bar{c}}\right)^{\bar{e}} = \sum_{\bar{e} \in \mathbb{E}} \tfrac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x}) \varepsilon^{\langle \bar{c}, \bar{e} \rangle}, \tag{2}$$

using $\bar{e}! := \prod_{j=1}^n e_j!$ and $\left(\varepsilon^{\bar{c}}\right)^{\bar{e}} := \prod_{j=1}^n \left(\varepsilon^{c_j}\right)^{e_j}$. Note that $t_i(\bar{x}(\varepsilon))$ is a polynomial in ε. Because $\prec$ is a total ordering on $\mathbb{E}$ and $\bar{c}$ induces $\prec$ it holds that $\langle \bar{c}, \bar{e} \rangle \neq \langle \bar{c}, \bar{f} \rangle$ for all $\bar{e}, \bar{f} \in \mathbb{E}$, $\bar{e} \neq \bar{f}$, i.e. no two terms of (2) have equal ε-exponents.

The value of $\lim_{\varepsilon \to 0^+} \operatorname{sgn}(A\varepsilon + B\varepsilon^2)$ is $\operatorname{sgn} A$ if $A \neq 0$, and is $\operatorname{sgn} B$ if $A = 0$. Generalizing this argument we conclude that the value of $\lim_{\varepsilon \to 0^+} \operatorname{sgn} \sum_{\bar{e} \in \mathbb{E}} \frac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x}) \varepsilon^{\langle \bar{c}, \bar{e} \rangle}$ is determined by that term of (2) that has the lowest ε-exponent among those with $\frac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x}) \neq 0$. But only $\operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x})$ has to be considered because $\frac{1}{\bar{e}!}$ is strictly positive.

Not all partial derivatives of t_i at $\bar{x}$ are equal to zero because t_i is not identical to zero (else it would be useless as a primitive). So at least one $\bar{s} \in \mathbb{E}$ exists with $\operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x}) \neq 0$. $\bar{e} \prec \bar{f}$ implies $\langle \bar{c}, \bar{e} \rangle < \langle \bar{c}, \bar{f} \rangle$ because $\bar{c}$ induces $\prec$, i.e. $\prec$ orders by increasing ε-exponent. Hence, proposition 2) is proven.

Now we show 1). The already proven 2) states that $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) \neq 0$. From Definition 3 follows that $\bar{0} \prec \bar{e}$, with $\bar{0} := (0, 0, \dots, 0)$. Because of this and 2) it holds that $t_i(\bar{x}) \neq 0$ implies $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \operatorname{sgn} t_i(\bar{x})$ (note that $\mathrm{D}^{\bar{0}} t_i(\bar{x}) = t_i(\bar{x})$). Hence, the perturbation scheme is valid. □

Note that it is not necessary to know the value of $\bar{c}$ to compute the sign of the perturbed primitive. After an admissible ordering has been chosen, only partial derivatives must be computed with the original data as input until one not equal to zero is encountered. The sign of the derivatives has to be computed correctly to ensure a correct result of the algorithm. An integer or rational arithmetic with arbitrary precision (short: exact arithmetic) can be used to achieve this. But these arithmetics are slow. To speed up computations, interval arithmetic is used first (lazy arithmetic [1]). If a computed interval does not contain zero, the correct sign of its associated expression is known. Only if zero is contained in the interval the more costly exact arithmetic has to be used.

Now two orderings will be shown to be admissible: Lexicographic ordering $\prec_L$ (string comparison) is well known. Total degree ordering $\prec_T$ has the sum of all elements as its primary ordering criterion; lexicographic ordering applies for elements with equal sum.

Theorem 3 *Lexicographic and total degree ordering are admissible orderings.*

Proof: Both are obviously total orderings on $\mathbb{E}$. Let $\bar{c}_L = (\lambda^0, \lambda^1, \dots, \lambda^{(n-1)})$ for any $\lambda > G$. Because of $\langle \bar{c}_L, \bar{e} \rangle = \sum_{j=1}^n e_j \lambda^{j-1}$ for any $\bar{e} \in \mathbb{E}$, $\bar{e}$ can be seen as a digit string with base λ representing the number $\langle \bar{c}_L, \bar{e} \rangle$. Hence, $\langle \bar{c}_L, \bar{e} \rangle < \langle \bar{c}_L, \bar{f} \rangle$ if $\bar{e} \prec_L \bar{f}$ and vice versa.

Let $\bar{c}_T = \lambda^n + (\lambda^0, \lambda^1, \dots, \lambda^{(n-1)})$ for any $\lambda > G$. Then $\langle \bar{c}_T, \bar{e} \rangle = \sum_{j=1}^n e_j (\lambda^n + \lambda^{j-1}) = |\bar{e}| \lambda^n + \langle \bar{c}_L, \bar{e} \rangle$ for any $\bar{e} \in \mathbb{E}$. The same digit argument as above holds. But there is an additional most significant "digit" $|\bar{e}|$. So $\bar{c}_T$ induces $\prec_T$. □

4.1 A C++ perturbation library

This perturbation scheme for rational functions was implemented at our institute as a C++ library [5]. This library provides classes and operators so that expressions implementing topological primitives can be programmed in a style that is very close to the mathematical notation. For example the expression $\frac{x1-x2}{x3}$ is represented by the code fragment `(x1-x2)/x3`. Testing this code expression against zero (e.g. `(x1-x2)/x3 < 0`) handles the whole perturbation. The objects `x1`, `x2`, and `x3` are allowed to represent not only numerical values but also other expressions. This results in great flexibility, as for example a program function that expects a point works also if this point is computed from the intersection of two lines. So a single function of a program implements many different topological primitives.

Internally, the expressions are stored as a directed acyclic graph. The partial derivatives are computed symbolically from these representations as they are needed. Also hidden from the programmer is the use of libraries for interval and exact rational arithmetic that guarantee correct and efficient computation of signs. See [5] for details.

5 Influence of perturbation on the result of an algorithm

A valid perturbation scheme does not change the outcome of an algorithm if its input data contains no degenerate cases (see Definition 2). Now we will look at some problems and show how they respond to perturbation in the presence of degenerate cases.

Problems whose result depends continuously on their input data everywhere (e.g. the area of the convex hull of a point-set) are not affected by perturbation, i.e. the result for perturbed and original data is always the same. Yet, perturbation is useful here, because the algorithms can introduce degeneracies that are not related to the result. If, for example, a vertical sweep line is used, it is a special case if two points have the same x-coordinate. However, this does not affect the algorithm's result.

For other problems the result for the original data can be reconstructed from the result for the perturbed data. Regard the following problem: Given the point-set S, the set $R \subseteq S$ of real vertices[1] of the convex hull C of S is sought. Let $R(\varepsilon)$ denote the result of this problem for the perturbed point-set $S(\varepsilon)$. $C(\varepsilon)$ denotes the convex hull of $S(\varepsilon)$. Points that lie on the boundary of C while not being real vertices are sensitive to perturbation. If the perturbation moves such a point to the inside of C, it will not be a vertex of $C(\varepsilon)$, therefore it will not be contained in $R(\varepsilon)$. But if it is moved to the outside of C, it becomes a real vertex of $C(\varepsilon)$, i.e. it will be contained in $R(\varepsilon)$. In practice, the direction of movement is not foreseeable or controllable by the user.

Real vertices of S are always real vertices of $S(\varepsilon)$. Points of S that lie inside the convex hull will also lie inside the perturbed set's convex hull. Hence, $R \subseteq R(\varepsilon)$ and R can be recovered easily from $R(\varepsilon)$ by removing those points that lie on the convex hull's boundary but are not real vertices.

For some problems, the result for the original data can not be reconstructed from the result for the perturbed data. A modified version of the above example illustrates this: Given S, a set of points. The set $A \subseteq S$ of *all* points of S on the boundary of the convex hull of S is sought. Let $A(\varepsilon)$ denote the result of this problem for the perturbed point-set $S(\varepsilon)$. From the discussion above we can see that $A(\varepsilon) \subseteq A$: Any point $p \in A$ that is not a real vertex may be moved to the inside of the convex hull by the perturbation, i.e. $p \notin A(\varepsilon)$. Hence, in general $A(\varepsilon)$ does not contain enough information to reconstruct A.

[1] A vertex of the convex hull is called real, if there is no non degenerate line segment in the convex hull that contains this vertex in its relative interior.

6 A problem-specific perturbation scheme

For a given algorithm and input data let R and $R(\varepsilon)$ denote the results for the unperturbed and perturbed data, respectively. In the previous section we have shown that some problems will respond to the perturbation such that $R \nsubseteq R(\varepsilon)$. Thus, it is not possible to reconstruct the original solution from the perturbed solution. Now we will present a new perturbation method that can ensure the reconstructibility of R from $R(\varepsilon)$ by enforcing $R \subseteq R(\varepsilon)$.

The basic idea is to apply a specialized *problem-specific perturbation* to the input data, so that some degenerate cases are transformed into general cases in a controlled way. For the exemplary problem of the previous section one would like the problem-specific perturbation to ensure that all points that lie on the convex hull's boundary are real vertices. This can be achieved by perturbing all points away from an interior point of the convex hull. (An interior point can be constructed easily from any three points of the given point-set.)

To achieve this, a linear perturbation $\bar{x} + \varepsilon\bar{d}$ with $\bar{d} \in \mathbb{R}^n$, $\varepsilon\bar{d} := (d_1\varepsilon, d_2\varepsilon, \ldots, d_n\varepsilon)$ suffices, because only the direction of the perturbation is relevant. $\bar{d}$ has to be chosen according to the problem at hand and the actual value of $\bar{x}$.

This linear perturbation is not capable of removing all degeneracies from the data, because it is controlled by the user. Therefore the perturbation $\varepsilon^{\bar{c}}$ of Section 4 has to be applied, too; we will call it *general perturbation* from now on. Combining both generates the problem-specific perturbation $\bar{x} + \varepsilon\bar{d} + \varepsilon^{\bar{c}}$. Clearly the general perturbation must not change the effect of the linear perturbation. This is achieved by choosing sufficiently large values for the components of $\bar{c}$. Informally speaking, $\varepsilon^{\bar{c}}$ has to be much smaller than $\varepsilon\bar{d}$.

Theorem 4 *For every admissible ordering $\prec$ there exists a $\bar{c} \in \mathbb{N}_+^n$ such that for every $\bar{d} \in \mathbb{R}^n$ the following statements are true:*

1) $\bar{x}(\varepsilon) = \bar{x} + \varepsilon\bar{d} + \varepsilon^{\bar{c}}$ is a valid perturbation scheme.

2) For every primitive t_i and every $\bar{x} \in \mathbb{R}^n$ there is a $\bar{s} \in \mathbb{E}$ with $\lim_{\varepsilon\to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x} + \varepsilon\bar{d}) \neq 0$ and $\lim_{\varepsilon\to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon\bar{d}) = 0$ for all $\bar{e} \in \mathbb{E}$, $\bar{e} \prec \bar{s}$, such that

$$\lim_{\varepsilon\to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \lim_{\varepsilon\to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x} + \varepsilon\bar{d}).$$

3) The linear perturbation takes precedence over the polynomial perturbation, i.e. for every primitive t_i and every $\bar{x} \in \mathbb{R}^n$:

$$\lim_{\varepsilon\to 0^+} \operatorname{sgn} t_i(\bar{x} + \varepsilon\bar{d}) \neq 0 \implies \lim_{\varepsilon\to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \lim_{\varepsilon\to 0^+} \operatorname{sgn} t_i(\bar{x} + \varepsilon\bar{d}).$$

Proof: Let $\bar{r} \in \mathbb{N}_+^n$ induce $\prec$ according to Definition 3. T is the maximum total degree over all primitives. We will show that with $\bar{c} = (T+1)\bar{r}$ the properties stated in the theorem hold. Let $i \in \{1, 2, \ldots k\}$ and $x \in \mathbb{R}^n$ be fixed but arbitrary.

Firstly we show 2). Because t_i is a continuous function and $\operatorname{sgn}(u)$ is continuous at $u \neq 0$,

$$\mathrm{D}^{\bar{e}} t_i(\bar{x}) \neq 0 \implies \lim_{\varepsilon\to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon\bar{d}) = \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x}) \qquad \forall\, \bar{e} \in \mathbb{E} \qquad (3)$$

holds. From this and the fact that not all partial derivatives of t_i at $\bar{x}$ are equal to zero we get

$$\exists\, \bar{s} \in \mathbb{E} : \lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}) \neq 0. \tag{4}$$

Applying the Taylor formula to $t_i(\bar{x}(\varepsilon))$ yields

$$t_i(\bar{x} + \varepsilon \bar{d} + \varepsilon^{\bar{c}}) = \sum_{\bar{e} \in \mathbb{E}} \tfrac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}) \varepsilon^{\langle \bar{c}, \bar{e} \rangle}. \tag{5}$$

$\mathrm{D}^{\bar{e}} t_i(\bar{x}+\varepsilon\bar{d})$ is a polynomial in ε of degree less than or equal to T for every $\bar{e} \in \mathbb{E}$. Because of this and the fact that $\langle \bar{c}, \bar{f} \rangle - \langle \bar{c}, \bar{e} \rangle > T$ for all $\bar{e} \prec \bar{f}$ no two terms of (5) have an ε-exponent in common. Therefore

$$\lim_{\varepsilon \to 0^+} \operatorname{sgn} \left(\tfrac{1}{\bar{e}!} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}) \varepsilon^{\langle \bar{c}, \bar{e} \rangle} + \tfrac{1}{\bar{f}!} \mathrm{D}^{\bar{f}} t_i(\bar{x} + \varepsilon \bar{d}) \varepsilon^{\langle \bar{c}, \bar{f} \rangle} \right) =$$

$$\begin{cases} \lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}) & \text{if } \lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}) \neq 0 \\ \lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{f}} t_i(\bar{x} + \varepsilon \bar{d}) & \text{else} \end{cases}$$

holds for all $\bar{e} \prec \bar{f}$. Generalizing this argument we get an analogous result for all terms of (5), i.e. for $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon))$. This and (4) proves 2).

Now we show 1). The already proven 2) states that $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) \neq 0$. From 2) and (3) follows that $\operatorname{sgn} t_i(\bar{x}) \neq 0$ implies $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon)) = \operatorname{sgn} t_i(\bar{x})$ because $\bar{0} \prec \bar{e}$ for all $\bar{e} \in \mathbb{E}$ (note that $\mathrm{D}^{\bar{0}} t_i(\bar{x}) = t_i(\bar{x})$). Hence, the perturbation scheme is valid.

Finally we show 3). From 2) and $\bar{0} \prec \bar{e}$ for all $\bar{e} \in \mathbb{E}$ follows that $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}+\varepsilon\bar{d})$ (if it is not zero) is the value of $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x}(\varepsilon))$. Hence, the linear perturbation takes precedence over the polynomial perturbation. $\square$

Note the following: for general perturbation (see Theorem 2) $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x} + \varepsilon^{\bar{c}}) = \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x})$ for some $\bar{s} \in \mathbb{E}$ holds; for problem-specific perturbation (see Theorem 4) $\lim_{\varepsilon \to 0^+} \operatorname{sgn} t_i(\bar{x} + \varepsilon \bar{d} + \varepsilon^{\bar{c}}) = \lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x} + \varepsilon \bar{d})$ holds. This is the *only* difference regarding the evaluation of perturbed primitives for the two perturbations.

The value of $\lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d})$, for $\bar{e} \in \mathbb{E}$ and $\bar{x}, \bar{d} \in \mathbb{R}^n$ can be computed using polynomial interpolation. Let t_i be a multivariate polynomial of total degree T. Then $\mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}))$ is a polynomial in ε of maximal degree T. Computing the value of $\mathrm{D}^{\bar{e}} t_i(\bar{x} + \varepsilon \bar{d}))$ for $\varepsilon = 0, 1, 2, \ldots, T$ allows to determine its polynomial coefficients.

The perturbation library mentioned in Section 4.1 can easily be extended to handle problem-specific perturbations. Only the evaluation of the partial derivatives has to be changed to the computation of $\lim_{\varepsilon \to 0^+} \operatorname{sgn} \mathrm{D}^{\bar{s}} t_i(\bar{x}+\varepsilon\bar{d})$ according to method stated above.

6.1 A problem-specific perturbation for line segment intersection

Given a set of line segments, each one specified by the coordinates of its end points, a list of all intersecting segments is sought. The algorithm of Bentley and Ottmann as described in [2] can solve this problem when the intersection of any two segments is either empty or a single point and no degeneracies are present in the data. Examples for these degeneracies are: two points (i.e. end points or intersection points) are vertically aligned (a vertical sweep line is used); two points are coincident.

General perturbation ensures these conditions and allows the implementation of the algorithm as presented in the paper. But as we can see in Fig. 1, cases where the intersection contains the end point of a participating segment may be affected by the perturbation. (Note that this also includes intersecting segments that are collinear.) So perturbation might result in the loss of parts of the result list.

We construct a linear perturbation to prevent this loss. The first step is to choose $\bar{d}$ such that the segments are extended by it. Two entries of $\bar{d}$ belong to each end point and build its *linear perturbation vector*. This vector is parallel to the matching segment and is directed to point away from the segment's other end point. The length of the perturbation vector is irrelevant, but it must not be zero.

With such a linear perturbation, the segments surely intersect after a general perturbation for the case presented in Fig. 1 (see Fig. 2). If it is certain that no collinear

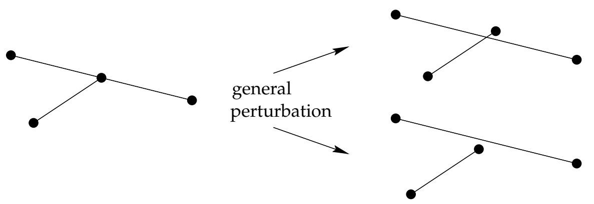

Figure 1: Possible loss of intersection via general perturbation.

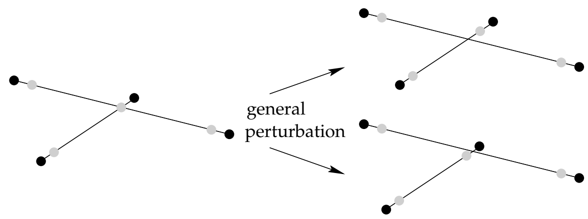

Figure 2: Linear perturbation prevents loss of intersection (original data is grey).

segments intersect, we are done. Any algorithm can then be used to solve the problem, as long as it works for the general intersection case, where the intersecting point of two segments does not coincide with any end point of the participating segments. The problem-specific perturbation (which includes the general perturbation) ensures that no degenerate case ever occurs, and that no intersection is lost.

A little more has to be done if collinear intersecting segments can not be ruled out. Then it is necessary to split every segment into two perturbed segments as shown in Fig. 3. The perturbation vectors for the end points of a segment are all of same length and have an angle of 45 degrees in respect to their segment. The two perturbed segments are tagged with an index so their unperturbed ancestor can be found out later.

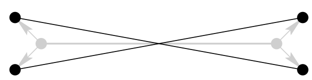

Figure 3: Linear perturbation to preserve all intersections.

When the perturbation vectors are chosen properly the following holds: If two unperturbed segments intersect in any form, then at least one pair of the matching perturbed segments intersect. This holds even for degenerate segments of length zero, if an arbitrary direction is chosen to perturb them. The unperturbed result can be reconstructed easily from the resulting list of intersecting pairs of perturbed segments by removing the superfluous intersection pairs. These are pairs that have the same unperturbed ancestor and pairs whose ancestor pairs appear more than once in the result list.

The following condition suffices to ensure the property just stated: All the perturbation vectors of end points of parallel line segments have to be of equal length. This result is obtained in the following way: Given two collinear intersecting line segments, a condition of the form $\lim_{\varepsilon \to 0^+} \operatorname{sgn} E > 0$ is formulated that, if fulfilled, guarantees the intersection of the two perturbed segments. The relevant perturbation vectors are variables of the expression E.

In practice, the perturbation vectors are determined like this. Let S be a segment with end points $P = (P_x, P_y)$ and Q. $V = P - Q$ is a vector with the length of S and direction from Q to P (set V to an arbitrary non–zero value if $P = Q$, e.g. $V =$

$(1,0)$). $V^{\perp} := (-V_x, V_y)$ is a vector with same length as V, but perpendicular to it. The four perturbation vectors of S are established by the expressions $\pm\dfrac{V \pm V^{\perp}}{\max(V_x, V_y)}$. The normalization with $\max(V_x, V_y)$ ensures two things. Firstly, the perturbation vectors of all parallel segments are of the same length. Secondly, in contrast to normalization with the length of V, the perturbation vector's coordinates are rational numbers, provided the coordinates of the end points are rational numbers. This is important to allow the use of an exact rational arithmetic to compute the signs of perturbed primitives correctly.

7 Conclusion and future work

A general symbolic perturbation scheme was presented that removes the need of handling degeneracies in geometric algorithms. The derivation given is considerably easier than comparable ones given elsewhere, and it allows the topological primitives to be rational functions. It has been shown in principle how this perturbation method can be implemented as a C++ library using a combination of interval and exact rational arithmetic for computations.

From the investigation of the influence of perturbation on the solution of geometric problems we learned that it is not possible to reconstruct the correct solution for all geometric problems when perturbation is applied to them. We presented a novel approach to overcome this deficiency with problem-specific perturbations. We showed that the perturbation library can easily be adapted accordingly. This has yet do be done. A problem-specific perturbation for the intersecting line segments problem was defined.

More geometric problems need to be investigated regarding the influence of perturbation and the applicability of problem-specific perturbation to preserve their solutions. Experiments have to be carried out to obtain information about the practical performance of the method.

References

[1] M. O. Benouamer, et al. A lazy solution to imprecision in computational geometry. In *Proc. 5th Canadian Conf. Comp. Geometry*, p. 73–78, Waterloo, Canada, 1993.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[3] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[4] I. Emiris and J. Canny. A general approach to removing degeneracies. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, p. 405–413, 1991.

[5] W. Pitzeier. *Implementing Geometric Algorithms Considering All Special Cases.* Doctoral thesis, Vienna University of Technology, Inst. of Computer Graphics, Vienna, Austria, 1997.

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

[7] R. Seidel. The nature and meaning of perturbations in geometric computing. In *Proc. 11th Sympos. Theoret. Aspects Comput. Sci.*, volume 775 of *LNCS*, p. 3–17. Springer-Verlag, 1994.

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

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