

Exact Numerical Perturbation *

Koji Ouchi John Keyser
kouchi@cs.tamu.edu keyser@cs.tamu.edu
Department of Computer Science
3112 Texas A&M University
College Station, TX
77843-3112

Technical Report 2005-01-2

January 2, 2005

Abstract

We present our exact numerical perturbation technique for eliminating degeneracies occurring in geometric modeling processes. Consider a geometric modeler that performs a set of geometric operations (e.g. CSG-based Boolean operations). We would like to make the geometric modeler “robust” with respect to degeneracies, meaning that it will not crash.

Our approach achieves many of the same results as symbolic perturbation, but requires very little algorithmic adjustment. Our perturbation technique differs significantly from the other methods (such as fixed-precision perturbation) in the following ways:

First, exact computation is used in order to avoid problems caused by numerical errors. Unlike the methods using fixed precision arithmetic or interval arithmetic, there is no need to keep track of errors

*This work funded in part by NSF awards DMS-0138446 and CCR-0220047

together with the associated geometric objects. Exact computation allows us to choose an arbitrarily small level of perturbation, allowing us to mimic the theoretical results of symbolic perturbation, unlike fixed-precision approaches. Exact computation allows the theoretical algorithms to be implemented straightforwardly. In particular, degeneracy detection can be done correctly.

Second, unlike symbolic perturbation, our perturbation scheme actually modifies the surfaces of solids. Both the direction and the amount of a perturbation must be determined exactly. The solids produced by our approach may differ from the input solid. However, the resulting output solid does not have any inherent degeneracies, such as edges of length zero or surfaces of area zero.

Besides giving a detailed description of degeneracies, how our approach handles them, and how it compares with other approaches, we refer to some simple examples to show that it can be used successfully on real applications.

1 Introduction

Robustness is a well-known problem in geometric and solid modeling. From an algorithm or approach design point of view (as opposed to a computer hardware or programming error point of view), all robustness problems arise from making an invalid assumption in the creation of an algorithm. One common example of such an assumption is that computation follows the Real RAM model [3], where one assumes that all arithmetic operations are exact over the real numbers, and in unit time; in practice this is not achieved.

Our concern here is with a separate assumption, the general position assumption. Stated roughly, this assumption says that small changes in the input will not change the nature of the output. More specifically, for geometric modeling this can be thought of as an assumption that small changes in the input change the numerical (or geometric) aspects of the output, but not the combinatorial (or topological) aspects. A violation of this assumption is referred to as a degeneracy.

Unfortunately, like violation of the Real RAM model assumption, degeneracies are common in many real-world applications. Sometimes these problems are unintentional, for example due to round-off error in computation, low sampling frequency, or poor training of a designer. Other times the problems may be intentional, for example a designer placing two objects next

to each other, touching only at a point. Regardless of the source, however, solid modeling software must be able to deal with degeneracies.

In this paper, we propose an *exact numerical perturbation* technique for dealing with degeneracies, and thus achieving more robust geometric operations. We discuss the ways in which this approach varies from other approaches to dealing with degeneracies, and reasons why we believe it can be superior in certain situations. This paper generalizes several aspects presented in preliminary form in our short paper at last year's Solid Modeling conference [22].

1.1 Main Result

The main result we present here is a formal description of the exact numerical perturbation approach for removing degeneracies. We describe the prerequisites for the approach, why the approach works, and how it can be implemented in practice. We also present a detailed description of our application of the approach to examples from solid modeling applications.

1.2 Organization

The rest of this paper is organized as follows. Section 2 discusses in greater detail the nature of degeneracies, particularly in solid modeling applications. In section 3, we give a brief summary of related work, and review alternative approaches for handling degeneracies by comparing our exact numerical perturbation technique to them. In section 4, we give a more formal description of our proposed global numerical perturbation method, including a discussion of how it should be implemented. In section 5, we show examples of exact numerical perturbation applied to resolve degeneracies from some solid modeling examples. Finally, we conclude in section 6.

2 Nature of Degeneracies

We discuss here the meaning of degeneracies, and how these manifest themselves in solid modeling.

2.1 Types of Degeneracies

The term “degeneracy” can have several different meanings depending on its context. The most general definition has to do with a description of program flow. As a program is executed, branching decisions are made on the basis of various *predicates*, where the code branches one way when the predicate is < 0 , and another way when it is > 0 [23] [25]. Regardless of how values $= 0$ are handled, situations that lead to a predicate being evaluated to 0 are considered degenerate. An infinitesimally tiny perturbation in the data could change the evaluation of the predicate to be either positive or negative, and thus change program flow.

If programs correctly account for all predicates, i.e. handle every case where a predicate evaluates to 0 consistently and ensure that there is no error in the evaluation of the predicates, they are considered robust. Exact computation ensures that predicates are evaluated without error; there are a few different approaches for handling the cases where the evaluation is $= 0$.

This also means that degeneracies are program-specific. A situation degenerate for one program might not be degenerate for another, if that other program does not encounter the same predicate (e.g. because it uses a different algorithm). However, there are other situations such that any algorithm performing a particular task will encounter a degeneracy. As a simple example, consider a collision-detection test on two spheres touching at a single point. Regardless of the collision detection test used, a degeneracy will be encountered since a small change in the position of a sphere will make the spheres either interpenetrate or separate, and this will appear as a degenerate situation for some predicate within any collision detection program.

We thus choose to divide degenerate situations into separate categories: *Input* degeneracies are those that are fundamental to the input data itself, and will be encountered in any geometric program. *Unpredictable* degeneracies are those due solely to arbitrary choices of the program. Finally, we name a third category, *intentional* degeneracies, which are situations generated within a program (rather than input to the program), and which the program later relies on existing (i.e. it assumes that they are true at later points). An example of an intentional degeneracy is a program generating the midpoint of two points, then relying on the fact that it is a midpoint at a later stage.

Because unpredictable degeneracies are program-specific, they must be dealt with in terms of general programming approaches; an entire separate document could be written to discuss design and programming methodolo-

gies to reduce problems with unintentional degeneracies. Thus, we will not deal with them directly, here, other than to note that our proposed method can easily be adapted to handle many of these unpredictable degeneracies. Likewise, we will not directly address intentional degeneracies. Our main goal with them is to ensure that intentionally degenerate situations remain so throughout the program. So, our focus is on how to handle the input degeneracies, and specifically those encountered in geometric and solid modeling. Unless specified otherwise, all future references to degeneracies thus refer to input degeneracies.

2.2 Degeneracies in Solid Modeling

In geometric and solid modeling, degeneracies can be thought of as those for which a small (i.e. infinitesimal) change in the input data will change the nature (i.e. the required topological or combinatorial structure) of the output. As an example, consider evaluating a Boolean operation on two solids. If the two solids meet only at a point, this is a degenerate situation: any slight perturbation in one direction and they will intersect in a volume, while any slight perturbation in another direction and they will not intersect at all.

Degenerate cases are common within solid modeling. For a boundary evaluation example, figure 1 shows several examples of degenerate configurations. Similar examples easily arise in parametric design evaluation; for example, a hole in an object might be specified with radius such that it intersects another boundary of the object at a point, as in figure 2. Most solid modeling applications requiring numerical computation will encounter similar cases.

2.3 Handling Degeneracies

Unfortunately, it is not always clear exactly what should be done to handle degenerate situations. One is tempted to consider the “ideal” result to be one in which the result of the degenerate situation can be output exactly as it is given. Unfortunately, this often creates problems of its own, in that such an exact representation may be far more complex than desired. One example comes from the well-known fact that regularized Boolean operations on manifold solids do not necessarily yield a manifold result. If we are willing

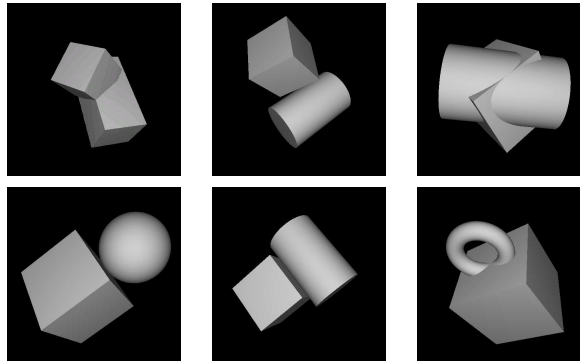


Figure 1: Examples of degenerate configurations appearing in boundary-evaluations for solid modeling.

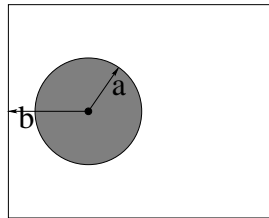


Figure 2: An example of a potential degenerate configuration (when $a = b$) from parametric design.

to represent only manifold models, then the “exact” result cannot be represented. Trying to represent the exact result, such as with a non-manifold data structure in the example, can drastically increase the complexity of the computation, for all cases.

A second approach is to represent the output as a “combination” of results, i.e. as a superposition of the possible perturbed values. This is very attractive, particularly in the sense that input data may be given with uncertainty, and thus the output can reflect that uncertainty. Unfortunately, this approach often translates numerical uncertainty (which is easily understood) into combinatorial uncertainty. It can be very difficult to understand and interpret such combinatorial uncertainty. This can be particularly a problem as computation is iterated, i.e. the output of one operation becomes input to the next. Handling such uncertain data as input can become very complex,

and rather than adding understanding, such uncertain output can quickly seem meaningless.

Finally, another approach is to choose to interpret the data as one of the “nearby” non-degenerate situations. There are several variations on this approach, and perturbation schemes (including ours) fall into this category. The benefit to this approach is that the output data in degenerate situations falls into the same general categories as for any generic input. The key drawback (beyond any additional computational complexity) is that the problem solved is not the one specified. Also, choosing how to interpret the data can become extremely important—if the “interpretation” of the data is poor, the result might not be the one desired (or even close to the one desired).

3 Background and Overview

We begin this section with a brief review of related work, in section 3.1. In section 3.3 we review previous approaches to dealing with degeneracies, and compare them with our approach. To enable better comparison, we first provide a brief overview of our proposed approach, in section 3.2.

3.1 Previous Work

Dealing with issues of robustness has been an active area of work for quite some time, now. Much of the need for robustness was highlighted by the work of Hoffmann et al. [17]. Work on robustness issues has continued, particularly within the computational geometry community.

Exact computation [18] [2] as a method for dealing with numerical error has been addressed within the solid modeling community. Much of the earliest work focused on polyhedral solids, with only limited work on curved solids. Among the work on exact computation in solid modeling is that of Sugihara and Iri [28], Yu [30], Benouamer et al. [1], Sugihara [27], and Fortune [10]. The work presented here builds off of earlier work on exact solid modeling [20] [19]. More general work supporting exact computation includes the development of the LEDA [21] and CORE [18] libraries, along with the more general algorithms supported by CGAL [8].

For dealing with degeneracies, special-case code has been the predominant approach used. Examples of special-case approaches can be seen in [16]. For curved solids, the issues of degeneracies become more complicated. A

great deal of effort has focused just on handling the intersections of quadric surfaces, such as that of Farouki et al. [9] and Geismann et al. [12].

Perturbation methods have arisen as a more general way of dealing with degenerate situations. In a perturbation method, we modify the input data and perform the computation over the perturbed input data which has no degeneracies. There are two types of perturbation methods; symbolic and numerical. In symbolic perturbation, we perturb the input data by symbolic amounts, perform the computations, and compute the final answer as the limit as the symbolic amount approaches 0. On the other hand, in numerical perturbation, we actually perturb the input data by some amounts and perform the computation. The adjective numerical is meant to be something which is not symbolic. A common misunderstanding (addressed in our work here) is that numerical perturbation does not involve exact computation; actually exact computation offers several benefits when coupled with numerical perturbation.

The major drawback of symbolic perturbation is that the schemes must either use symbolic computation, which is extremely expensive, or define every predicate directly from the input, which is impractical for large programs. The earliest perturbation scheme was probably that of Edelsbrunner and Mücke [5]. Emir and Canny varied the perturbation used to a simpler one, and applied it to a wider variety of cases [6] [7]. Yap provided an even more generalized perturbation approach [29]. Seidel provides a summary of these techniques, along with a critique of the general perturbation scheme itself [25]. For solid modeling applications, Fortune actually describes the use of symbolic perturbation for linear solids [10]. An example of our proposed approach applied to one case was also presented recently [22].

There are also several perturbation schemes that rely on fixed-precision computation [13] [11] [14] [15] [24]. In some of these approaches (e.g. [14]), geometric objects are fattened (e.g. points become circles) and some adjustments have to be made to the geometric algorithms.

3.2 Exact Numerical Perturbation

Although the precise definition of exact numerical perturbation will be provided in section 4, we give a brief descriptive overview here. We will consider each of the three words in the name.

First, we use a *perturbation* scheme. That is, we deal with the degeneracies by solving a slightly perturbed version of the input problem. The basic

idea is that the perturbed version of the problem is “close” to the original problem, and thus the results that we obtain are either acceptable or easily converted into something acceptable as an answer.

Second, we use a *numerical* perturbation approach. That is, we make actual changes in the numerical values of the data in order to apply our perturbation. This is in contrast to the more commonly known symbolic perturbation schemes, where the input data is perturbed by an infinitesimal amount that is never explicitly represented as a number.

Finally, we use an *exact* computation framework for our approach. This means that we eliminate all problems with numerical error by using exact representations for all objects (maintaining whatever precision is necessary, so that no error is introduced), and exact numerical computations—computing with enough precision that any decisions to be made in a program are guaranteed to be correct. A few aspects of exact computation should be noted:

- Since numerical error can both create spurious degeneracies and remove desired degeneracies during the course of computation, it is important to have a good handle on numerical error before dealing with degeneracies; we do so by using exact computation.
- Using exact computation will lead to a significant decrease in efficiency, though the loss of efficiency may not be as bad as one might expect [19].
- Exact computation (enough precision to guarantee every decision is correct) differs from exact arithmetic (every computation is done to full precision). Exact arithmetic can be used to achieve exact computation, and might need to be used as a “last resort,” but it is not always necessary. Using techniques such as lazy evaluation and filtering can also allow exact computation, with much less computational cost.
- Although our perturbation can increase the bit-length of the inputs, this does not necessarily lead to increased computation time, due to filtering.

3.3 Comparison

There have been a variety of methods previously proposed for dealing with degeneracies or perturbing data. To put our method in context, we compare it to four such approaches here.

3.3.1 Special Cases

Special case code involves detecting each degenerate case and creating a special case line of computation to handle the degeneracy (presumably differently than if the predicate had been positive or negative). Generally, this is the approach used whenever one wishes to create output that maintains the degenerate situation or creates “ambiguous” output; it is less effective than other methods for the “interpret data one way” approach. We outlined above (see section 2.3) the various advantages and drawbacks of these alternative ways of dealing with degeneracies. Treatment by special cases requires a great deal of analysis and extra coding, and it is difficult to ensure that all degeneracies have been found, and if so, that they are all handled in the correct way. In contrast, exact numerical perturbation requires very little algorithmic change.

3.3.2 Symbolic Perturbation

With symbolic perturbation, the geometric aspects of the output are considered to be unchanged - only the topological information is different from that specified. This can lead to output that may be difficult to interpret or deal with, and may cause more robustness problems as operations are iterated. For example, the resulting solid may have edges of length zero, surfaces of area zero, and the like. In contrast, our approach actually modifies the geometric information. Our resulting solid is thus different from what a designer might have specified (though we can ensure it stays within some distance envelope, for example), but the solid has no such problematic geometry.

Also, symbolic perturbation schemes usually require (in order to have any reasonable efficiency) that each internal predicate can be expressed directly in terms of the input data. While this is feasible for some small-scale, simple programs, it is completely impractical for larger-scale or iterated programs. In such programs, the deep computation path and repeated use of generated data would make tying everything back to the original input nearly impossible.

One similarity between methods is that both rely on exact computation; exact computation is usually a prerequisite for symbolic perturbation.

3.3.3 Numerical Perturbation

When numerical perturbation has been applied in the past, e.g. as in Sugihara’s work [28], it is usually done in the context of fixed precision computation. That is, data is perturbed pointwise across a grid made up of representable points in fixed-precision space. This significantly limits the types and amounts of perturbation that can be applied; though it works for several cases, it is not as general in application as our proposed approach. In contrast to these earlier methods, our use of exact computation allows for a wider range of perturbation amounts. In fact, as will be shown below, our approach yields the full generality that could be obtained through symbolic perturbation.

Numerical perturbation is also sometimes used as a “hack” to force completion of a computation at a local level. That is, if a computation comes out to zero (or a problematic case such as a rank-deficient matrix), the input is slightly (randomly) modified so that recomputation is (hopefully) successful. Application in this manner falls in the realm of folklore, and is completely insufficient for our purposes. Most notably, such local perturbation is usually unable to guarantee consistency across the entire output, as is necessary in solid modeling. Our method applies the perturbation at a more global level. Also, this approach can easily lead to the disappearance of intentional degeneracies, which is not desired by any application.

3.3.4 Other Input Perturbation

Recently, a different type of perturbation approach has been proposed by Song et al. [26]. In this approach, input data (specifically, the parametric surfaces) are modified in such a way as to meet a particular constraint (specifically, intersecting on a certain curve). That is, the input is perturbed in a very specific way to ensure a specific outcome. In contrast, our method modifies the input data, but in a more random fashion, rather than insisting on achieving a particular result. In its current form, the approach of Song et al. is not applied to degenerate data, and in fact the specific algorithms are limited in the number of surfaces that can be so adjusted, but the method has potential for future application in such a direction.

4 Exact Numerical Perturbation

In this section we give a formal definition of exact numerical perturbation, and show how it fits into the general perturbation framework. In order to provide context, we first review the formal definition of symbolic perturbation schemes as discussed by Seidel [25] [7], in section 4.1. Then, we describe how the exact numerical perturbation falls into the same framework in section 4.2. In section 4.3 we discuss issues related to implementation of exact numerical perturbation, and in section 4.4 we discuss practical implementation details to achieve specific results.

4.1 Perturbation

As discussed before, a perturbation method is a general approach for eliminating degeneracies. When a perturbation is applied, the input data is modified and the computation is performed over the perturbed input data. Since the perturbation eliminates degeneracies, the computation does not have to deal with degeneracies, and thus terminates without any trouble.

Seidel [25] [7] gives a syntactic definition for perturbations, though his focus is specifically on symbolic perturbations. We follow his discussion here, to demonstrate that we can frame our exact numerical perturbation in a similar manner. The material in this section is basically a paraphrase of Seidel's work, with the following changes:

- We restrict our discussion to linear perturbations, as opposed to more general ones.
- Unlike Seidel, we do not assume that the input data is uniform in dimension.

We include this section in order to provide context and justification for the following sections.

Consider the problem Π from some input space \mathcal{I} to some output space \mathcal{O} . We assume that an input instance x consists of n vectors x_1, \dots, x_n where

$$x_i = (x_{i1}, \dots, x_{im_i}) \in \mathbb{R}^{m_i}, \quad i = 1, \dots, n.$$

Thus, the input space \mathcal{I} is the Cartesian product $\mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$. Also, we assume the output space \mathcal{O} is endowed with some topology.

For an input instance $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, define a *perturbation* of x to be a tuple $x(\epsilon) = (x_1(\epsilon), \dots, x_n(\epsilon))$ of rays $x_i(\epsilon)$ in \mathbb{R}^{m_i} starting at $x_i \in \mathbb{R}^{m_i}$ for $i = 1, \dots, n$. That is

$$\begin{aligned} x(\epsilon) &= (x_1(\epsilon_1), \dots, x_n(\epsilon_n)) \\ &= (x_1 + \epsilon_1 d_1, \dots, x_n + \epsilon_n d_n) \end{aligned} \quad (1)$$

where $\epsilon = (\epsilon_1, \dots, \epsilon_n) \in \mathbb{R}_{\geq 0}^n$ and $d_i = (d_{i1}, \dots, d_{im_i}) \in \mathbb{R}^{m_i} \setminus \{0\}$ is some direction, i.e., some non-zero vector in \mathbb{R}^{m_i} for $i = 1, \dots, n$.

A *perturbation scheme* X assigns a perturbation $x(\epsilon)$ to every input instance x in the input space.

Given a problem $\Pi : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$ and a perturbation scheme X , define a *perturbed problem* of Π to be the problem $\bar{\Pi}^X$ from the input space $\mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ to the output space \mathcal{O} such that

$$\bar{\Pi}^X(x) = \lim_{\epsilon \rightarrow 0^+} \Pi(x(\epsilon)), \quad \forall x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}. \quad (2)$$

We assume that a perturbation problem is well-defined, i.e., the limit in the right hand side of (2) exists. Often, the perturbation scheme X is clear from the context, so we just write $\bar{\Pi}$ for the perturbed problem.

So, we want a perturbation scheme that follows this well-defined perturbation and that is continuous on some superset of the domain of the original problem so that some (ideally all) degenerate input instances are dealt with. Furthermore, we hope the computation of the perturbed problem is easier (since degeneracies do not need to be handled) than the computation of the original problem. On an input x , we solve the perturbed problem $\bar{\Pi}(x)$ instead of the original problem $\Pi(x)$. In a symbolic scheme, we must then post-process the information obtained from the output of the perturbed problem $\bar{\Pi}(x)$ in order to recover the output of the original problem $\Pi(x)$. When the perturbation is continuous, though, such post-processing is unnecessary.

Computations of problems are modeled by *extended algebraic decision trees*. An extended algebraic decision tree is ternary tree. Each interior node v is associated with the test function $f_v : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathbb{R}$ and its (three) branches are labeled by -1 , 0 and 1 , respectively. Each leaf v is associated with the result function $g_v : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$.

Let T be an extended algebraic decision tree that models computation of a problem $\Pi : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$. On an input $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, the computation of $\Pi(x)$ is modeled by a traversal of T from the root to some leaf.

At each internal node v of T , the sign of the test function $s_v(x) = \text{sgn}(f_v(x))$ is evaluated, then the branch labeled by $s_v(x)$ is taken to one of its subtrees and the computation is continued in that subtree. When a leaf v of T is reached, $g_v(x)$ is evaluated and returned.

We assume that, for every internal node (or leaf) v of T , the test function f_v (or the result function g_v , respectively) is defined and continuous at all input instances x that reach v .

A perturbed problem $\bar{\Pi}$ on an input $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ is also computed by a traversal of T . At each internal node v of T , instead of $s_v(x) = \text{sgn}(f_v(x))$, $\bar{s}_v(x) = \lim_{\epsilon \rightarrow 0^+} \text{sgn}(f_v(x(\epsilon)))$ is evaluated and used to take the branch to the appropriate subtree. When a leaf v of T is reached, instead of $g_v(x)$, $\bar{g}_v(x) = \lim_{\epsilon \rightarrow 0^+} g_v(x(\epsilon))$ is evaluated and returned.

Let $f : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathbb{R}$ be a continuous function, and let $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$. A perturbation $x()$ of x is said to be *valid* for f iff $\lim_{\epsilon \rightarrow 0^+} \text{sgn}(f(x(\epsilon)))$ exists and is non-zero. A perturbation scheme is said to be *valid* for f iff, for all $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, $x()$ is valid.

Theorem 4.1 Seidel [25] [7]

Let T be an extended algebraic decision tree that computes a problem $\Pi : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$. Furthermore, let X be a valid perturbation scheme for all the test functions appearing in T . Then

1. A perturbed evaluation of T computes the perturbed problem $\bar{\Pi}^X$.
2. If Π is continuous at $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ then $\bar{\Pi}^X(x) = \Pi(x)$.
3. The above statements remain true even if some of (or all) the 0-branches of T are removed.

Let $f : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathbb{R}$ be a continuous function. Then, $S = f^{-1}(0)$ is a hypersurface formed by the zero set of f .

Let $x()$ be a perturbation of $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$. Suppose some open initial segment of the curve $x()$ does not intersect the hypersurface S , i.e., $\exists (\tau_1, \dots, \tau_n) \in \mathbb{R}_{\geq 0}^n$ such that

$$\begin{aligned} \epsilon &= (\epsilon_1, \dots, \epsilon_n) \in (0, \tau_1) \times \dots \times (0, \tau_n) \\ \Rightarrow f(x(\epsilon)) &= f(x_1(\epsilon_1), \dots, x_n(\epsilon_n)) \neq 0 \end{aligned}$$

Then, for all the points $x(\epsilon)$ on this initial open segment, the values $f(x(\epsilon))$ must have the same non-zero sign. This implies the limit $\lim_{\epsilon \rightarrow 0^+} \text{sgn}(f(x(\epsilon)))$ exists and is non-zero, i.e., $x()$ is valid for f .

Recall that $x()$ is a tuple of rays $x_i()$ in \mathbb{R}^{m_i} starting at x_i , i.e., $x_i(\epsilon_i) = x_i + \epsilon_i a_i$ where $\epsilon_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \mathbb{R}^{m_i} \setminus \{0\}$. The curve $x()$ is invalid for f because 1) S contains some open initial segments of the curve $x()$ or 2) S is not “well-behaved” meaning that a straight line intersects with S at infinitely many points. We ignore this second case, since it is not the case for the interesting test functions such as polynomials and rational functions.

A slightly more conservative version of 1) can be stated as follows: if, for $i = 1, \dots, n$, the projection of S to \mathbb{R}^{m_i} does not contain some open initial segments of the ray $x_i(\epsilon_i) = x_i + \epsilon_i a_i$ then $x()$ is valid. The conditions for the conservative version of 1) are satisfied almost always. Hence, for a fixed but arbitrary input $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, the set of all tuples of directions $a = (a_1, \dots, a_n) \in (\mathbb{R}^{m_1} \setminus \{0\}) \times \dots \times (\mathbb{R}^{m_n} \setminus \{0\})$ such that the perturbation $x(\epsilon) = (x_1 + \epsilon_1 a_1, \dots, x_n + \epsilon_n a_n)$ is invalid for f is of measure 0. In other words, even if a tuple of directions $a \in (\mathbb{R}^{m_1} \setminus \{0\}) \times \dots \times (\mathbb{R}^{m_n} \setminus \{0\})$ is chosen at random, with probability 1, the perturbation $x(\epsilon) = (x_1 + \epsilon_1 a_1, \dots, x_n + \epsilon_n a_n)$ is valid for f . The last statement holds not just for a single function, but also for any finite set of continuous functions whose zero sets are smooth.

Proposition 4.2 *Random perturbations work almost always.*

We say that S is *well-behaved* iff every straight line L is either contained in S or every bounded segment of L intersects with S in at most finitely many points. We say S is *scale-invariant* iff, $\forall \alpha \in \mathbb{R}$, $a \in S$ implies $\alpha a \in S$. For example, polynomials, rational functions and analytic functions are all well-behaved. Practically, all primitive test functions arising in geometric algorithms are scale-invariant.

Lemma 4.3 Seidel [25]

Let $f : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathbb{R}$ be continuous, well-behaved and scale-invariant, and let $a = (a_1, \dots, a_n) \in (\mathbb{R}^{m_1} \setminus \{0\}) \times \dots \times (\mathbb{R}^{m_n} \setminus \{0\})$ be a tuple of directions, i.e., a tuple of non-zero vectors in $\mathbb{R}^{m_1}, \dots, \mathbb{R}^{m_n}$, with $f(a) \neq 0$. Then, for every $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, the perturbation $x(\epsilon) = (x_1 + \epsilon_1 a_1, \dots, x_n + \epsilon_n a_n)$ is valid for f .

Theorem 4.4 Seidel [25]

Let T be an extended algebraic decision tree for a problem Π and F be the set of test functions in T . Assume that all $f \in F$ are well-behaved and scale-invariant.

If $a = (a_1, \dots, a_n) \in (\mathbb{R}^{m_1} \setminus \{0\}) \times \dots \times (\mathbb{R}^{m_n} \setminus \{0\})$ is a tuple of directions, i.e., a tuple of non-zero vectors in $\mathbb{R}^{m_1}, \dots, \mathbb{R}^{m_n}$ such that $f(a) \neq 0$ for all $f \in F$, then, for every $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, the perturbation $x(\epsilon) = (x_1 + \epsilon_1 a_1, \dots, x_n + \epsilon_n a_n)$ is valid for F .

The theorem is paraphrased as “if you know just one non-degenerate input, then you can use it for a valid linear perturbation for every possible input.”

However, explicitly constructing such a “non-degenerate” input can be a difficult problem (this is the subject of section 4.4).

It is important to note that exact computation is necessary for implementing a perturbation scheme since resolving degenerate cases relies on exact sign determination of test functions.

4.2 Numerical Perturbation

We now describe exact numerical perturbations.

In section 4.1 we saw that, in the perturbed computation of a problem, the test functions and the result functions are evaluated at a symbolically perturbed input instance and then the limit when the symbolic amount of perturbation approaches zero is taken. For our numerical perturbation scheme, the procedure is the same, except the limit is not taken. By the continuity condition, all the test functions are evaluated to have the same signs as long as the amount of perturbation is small enough and the direction of a perturbation is the same. The output of the result function becomes slightly “larger” since objects of measure 0 no longer exist. Thus, exact numerical perturbation can be viewed as a *backward-stable* operation: we compute the output of a perturbed problem of the original input.

For the same reasons as the case of symbolic perturbation, exact computation must be used. The adjective “numerical” is meant to convey that the perturbation is not symbolic. In order to emphasize the fact that exact computation is required for our numerical perturbation, we call it an *exact* numerical perturbation.

We now give a formal description of a numerical perturbation (or an exact numerical perturbation) which is analogous to a symbolic perturbation. Note that this description of a numerical perturbation yields a more general form than that of the (fixed precision) numerical perturbations of prior work.

Consider the problem Π from some input space $\mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ to some output space \mathcal{O} that is endowed with some topology.

For an input instance $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, define a *numerical perturbation* of x to be a tuple of vectors

$$\begin{aligned} x(\delta) &= (x_1(\delta_1), \dots, x_n(\delta_n)) \\ &= (x_1 + \delta_1 d_1, \dots, x_n + \delta_n d_n) \end{aligned} \quad (3)$$

for some $\delta = (\delta_1, \dots, \delta_n) \in \mathbb{R}_{\geq 0}^n$ where $d_i = (d_{i1}, \dots, d_{im_i}) \in \mathbb{R}^{m_i} \setminus \{0\}$ is some direction, i.e., a non-zero vector in \mathbb{R}^{m_i} for $i = 1, \dots, n$.

A *numerical perturbation scheme* X assigns a numerical perturbation $x()$ to every input instance x in the input space.

Given a problem $\Pi : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$ and a numerical perturbation scheme X , define a *numerically perturbed problem* of Π to be the problem $\tilde{\Pi}^X$ from the input space $\mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ to the output space \mathcal{O} such that

$$\tilde{\Pi}^X(x) = \Pi(x(\delta)), \quad \forall x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}. \quad (4)$$

A numerically perturbed problem $\tilde{\Pi}$ on an input $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ is computed by a traversal of an extended algebraic decision tree, T , for computing the original problem Π . At each internal node v of T , instead of $s_v(x) = \text{sgn}(f_v(x))$, $\tilde{s}_v(x) = \text{sgn}(f_v(x(\delta)))$ is evaluated and used to take the branch to the appropriate subtree. When a leaf v of T is reached, instead of $g_v(x)$, $\tilde{g}_v(x) = \text{sgn}(r_v(x(\delta)))$ is evaluated and returned.

Let $f : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathbb{R}$ be a continuous function, and let $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$. A numerical perturbation $x()$ of x is said to be *valid* for f iff $\text{sgn}(f(x(\delta)))$ is non-zero and $\forall \epsilon \in (0, \delta)$, $\text{sgn}(f(x(\epsilon)))$ is the same as $\text{sgn}(f(x(\delta)))$. A perturbation scheme is said to be *valid* for f iff, for all $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, $x()$ is valid.

Theorem 4.5 *Let T be an extended algebraic decision tree that computes a problem $\Pi : \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n} \rightarrow \mathcal{O}$. Furthermore, let X be a valid numerical perturbation scheme for all the test functions appearing in T . Then*

1. *A numerically perturbed evaluation of T computes the numerically perturbed problem $\tilde{\Pi}^X$.*
2. *$\lim_{\delta \rightarrow 0+} \tilde{\Pi}^X(x) = \bar{\Pi}^X(x)$. Furthermore, if Π is continuous at $x \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$ then $\lim_{\delta \rightarrow 0+} \tilde{\Pi}^X(x) = \Pi(x)$.*

3. The above statement remain true even if some of (or all) the 0-branches of T are removed.

Proof 1. Described in the above.

2. The first statement is immediate from the definition for valid numerical perturbations. The second statement follows from the first statement and theorem 4.1.
3. By the same argument used to show theorem 4.1 3).

By the above theorem, a valid numerically perturbed computation of a problem evaluates the test functions to the same sign as a valid symbolically perturbed computation. Thus, by the same arguments as proposition 4.2 and theorem 4.4, we can show the following:

Proposition 4.6 *Random numerical perturbations work almost always. More precisely, even if directions of perturbations are chosen at random, almost always, there exists an appropriate choice for the amount of perturbations so that a numerical perturbation scheme is valid.*

Theorem 4.7 *Let T be an extended algebraic decision tree for a problem Π and F be the set of test functions in T . Assume that all $f \in F$ are well-behaved and scale-invariant.*

If $a = (a_1, \dots, a_n) \in (\mathbb{R}^{m_1} \setminus \{0\}) \times \dots \times (\mathbb{R}^{m_n} \setminus \{0\})$ is a tuple of directions, i.e., a tuple of non-zero vectors in $\mathbb{R}^{m_1}, \dots, \mathbb{R}^{m_n}$ such that $f(a) \neq 0$ for all $f \in F$, then, for every $x = (x_1, \dots, x_n) \in \mathbb{R}^{m_1} \times \dots \times \mathbb{R}^{m_n}$, there exists a numerical perturbation $x(\epsilon) = (x_1 + \delta_1 a_1, \dots, x_n + \delta_n a_n)$ that is valid for F .

4.3 Issues of Implementation

We have seen that a symbolic or numerical perturbation is valid iff the perturbed computation of the problem does terminate and produce some output. This does not necessarily mean that the output of a valid perturbed computation is the same as or very similar to the ideal one. There are several issues that must be dealt with in order to move from the theory of exact numerical perturbation to a practical implementation.

4.3.1 Direction of Perturbation

We showed above that numerical perturbation was valid, in a sense, assuming that the direction of perturbation was acceptable and that the amount of perturbation was small enough. We will deal with amount below. The choice of a good direction of perturbation is a problem common to all forms of perturbation, including symbolic and fixed-precision numerical.

As was stated in theorem 4.7, valid perturbations will exist in most cases, and as stated in proposition 4.6, almost any random perturbation direction will work. However, not all such random perturbations are considered “good.” The choice of a “good” perturbation direction will depend heavily on the problem itself. We want to choose a perturbation direction that is likely to capture the intent of the designer. That is, we want to choose some nearby configuration of the input, while eliminating the computational problems with degeneracies, that is still “close” to what the designer of the input intended. For example, earlier work on boundary evaluation [28] [10], including our prior work [22], uses an expansion and contraction operation on basic primitives to achieve a “good” perturbation. Since an appropriate choice of direction is problem-dependent, we do not propose a single solution, but rather describe below (section 4.4) a couple of general categories for choosing directions.

Often, to try to capture design intent, a non-random perturbation direction is used. It is possible that this perturbation direction, then, might not remove all degeneracies. An example is seen in figure 3, where if both objects are perturbed through expansion and contraction (see section 4.4), the degeneracy will remain. It is therefore important to choose perturbation directions that are appropriate for your application.

4.3.2 Amount of Perturbation

In symbolic perturbation, the output of a perturbed computation is usually geometrically correct but slightly modified topologically. The perturbed computation may add some extra things to the output, but, in the end, they will be shrunken so that they are of measure 0. In numerical perturbation, the output of a perturbed computation is truly different than that of the original problem - instead of measure 0 structures, we have “very small but positive measure” structures. Strictly speaking, the intent of the designer is lost when numerical perturbation is applied. However, in many cases our

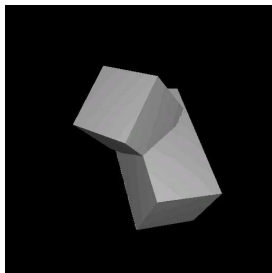


Figure 3: Example where perturbing inward and outward does not remove the degeneracy.

small structures will be no worse than the measure zero structures.

Another issue with the amount of perturbation comes from meeting the validity constraints of the above theorems. We have shown that exact numerical perturbation is valid if the amount of perturbation is “small enough,” i.e. so small that any smaller perturbation would not change the sign of any predicate. Unfortunately, determining a bound on δ that is “small enough” is not straightforward. In a practical sense, we want δ to be small enough to guarantee validity, but not so small as to create undue computational efficiency problems. This is a shortcoming of fixed precision numerical perturbation - you are limited in the bounds and values (as well as directions) for perturbation by the precision allowed. In contrast, we can always reduce our perturbation amount (making use of scale invariance) to any level desired.

One way of getting around this difficulty is to rely on an idea of a *global tolerance*. That is, you assume that the input geometric data is correct within some amount, τ . The global tolerance often is used to take in to account the inexact nature of the real-world data. Any perturbation of the input $\delta < \tau$ is allowed, i.e. the input geometric data can be perturbed as much as is necessary to allow the program to run, as long as the perturbation is less than τ . In some cases, this can be even more desirable than the precise , as it would allow perturbation of the input *anywhere* within the tolerance envelope, potentially *lowering* computational cost (if perturbed to, e.g. a lower bit-length) and computational complexity if perturbed to, e.g. a position where the output had lower combinatorial representation requirements). As the tolerance value approaches zero, though, the perturbed surfaces still approach the original surfaces, and thus we can achieve valid exact numerical

perturbation (as outlined above) if we can determine a good bound on δ .

4.3.3 Exact Computation

Our method requires implementation within an exact computation framework. Although this is a feasible approach [19], support for exact computation is not yet commonplace or widely accepted. It also involves a significant cost in efficiency.

Numerical perturbation has the potential to drastically increase the bit length of input data (it could also decrease it, though). For example, the number 1 might be perturbed to 1.000001 (or vice-versa) . Several exact computation techniques, though, tend to minimize the impact of this increase. Floating point filters are a clear example—usually the perturbed data will have the same filtered representation as the unperturbed, so there is only additional computation required when the filters fail. This should be particularly true in the continuous areas of the computation, so that additional time is spent only near the original degeneracies (when filters would be expected to fail).

It is important to note the reason we are using exact computation and handling degeneracies is to achieve greater robustness and consistency. The goal is not necessarily to have an exact solution to the input problem—in the real-world it is very rare that the output solids would need to be found with absolute precision. However, without accounting for numerical error and degeneracies, programs are subject to crashes or inconsistent output, neither of which is a desirable condition. We want to have a program that can reliably produce a set of valid output, for a wide variety of input.

4.4 Practical Implementation of Exact Numerical Perturbation

The problem now becomes how we can choose to implement exact numerical perturbation in a real-world setting. First, an exact computation approach is required for the computation [19].

Theorem 4.7 states that “if you know just one non-degenerate input, then you can use it for a valid linear perturbation for every possible input.” But, given a problem, explicitly constructing “non-degenerate” input is not obvious. Proposition 4.6, assures us that even if directions are chosen at random, there almost always exists a valid numerical perturbation scheme.

On the other hand, as discussed in the pervious section, we would like to maintain the designer’s intent.

Although the direction and amount of numerical perturbation to allow will be specific to an individual problem, we will describe here two simple and direct instantiations of numerical perturbation, “random translation” and “contraction/expansion.” These can be used in many geometric modeling situations. In practice, we will choose directions of perturbation according to a ceratin criterion, and then adjust amounts of perturbation (e.g. to meet a global tolerance). There are many other possible ways of choosing perturbation directions, e.g. via rotation or arbitrary affine transformation, or even by *local* translation of geometric elements; we mention these as just two possibilities.

4.4.1 Random Translation

Random translation is a numerical perturbation scheme where input instances are translated in random directions. In these instances, we pick the direction at random, and translate all geometric elements accordingly. This is easily adapted to a computed-direction translation. The random translation approach fits most easily into the description of exact numerical perturbation outlined above. With a truly random choice of direction, we can be assured of, with probability 1, eliminating any degeneracies.

We note, however, that this perturbation is less likely to meet any criteria of capturing designer’s intent. Even though the operation might complete, the output can be different from the what the designer intended. This is essentially due to the lack of consideration of the amount of perturbation. Proposition 4.6 assures only the existence of some numerical perturbation scheme but does not tell any information about how much an input will be perturbed. In symbolic perturbation, this is not a problem since the limit when this amount approaches to zero is taken. In numerical perturbation, however, an input is actually perturbed by some positive amount which might be larger than the tolerance value. So, we must somehow check whether or not the designer’s intent is maintained, which will have to be done analytically on a case-by-case basis.

4.4.2 Expansion / Contraction

A second numerical perturbation approach is expansion/ contraction. This has been shown to be quite useful in the handling of boundary evaluation operations. Sugihara and Iri proposed the principles early on [28], and Fortune later adapted them [10] for polyhedral solids. We have implemented exact numerical perturbation over curved solids, as described below and in an earlier paper [22].

With expansion/contraction, the input data is effectively scaled outward or inward, from an object centroid. For simple geometric objects (such as those encountered as CSG primitives), this is usually straightforward. By adjusting which objects are perturbed inward (contracted) and which outward (expanded), we can capture designer's intent relatively well.

Expansion / Contraction for Primitives Applying expansion/contraction to general curved surfaces is nontrivial. That is, it is easy to define such an operation in theory, but difficult to implement it in practice. Fortunately, however, it is relatively easy to apply expansion/contraction to the standard CSG primitives (boxes, generalized cones, ellipsoids, and tori). Perturbations at one level of a tree can always be propagated to the primitives. Details are described in section 5.2.

For each of these primitives, we scale the entire solid by a certain amount (bounded from above by the global tolerance), fixing its centroid. By fixing the centroid of a solid, we scale the solid outward/inward by the same amount in any direction. More precisely: Fix a point p on a surface of a solid and let us use the same symbol p for a position vector originated at the centroid of the solid. Define the expansion or contraction of p by δ if $q = (1 + \delta)p$ and $\delta > 0$ or $\delta < 0$, respectively. For example, an ellipsoid can be expanded just by increasing the radius vector.

There are a few shortcomings to the exact numerical expansion and contraction as applied to boundary evaluation, which we list here.

First, there are a limited number of cases for which expansion or contraction is not sufficient.

Second, it is possible with perturbation schemes to create small, unintended features in objects. While these will indeed be small, and should not affect the overall topology of the solid, they can be annoying to deal with in subsequent computation.

5 Examples of Numerical Perturbations



a

b

c

Figure 4: Real-world examples. a. Cargo hatch, b. Commander hatch, c. Engine

We see here some examples of degeneracies resolved by applications of the exact numerical perturbations.

5.1 Random Translation

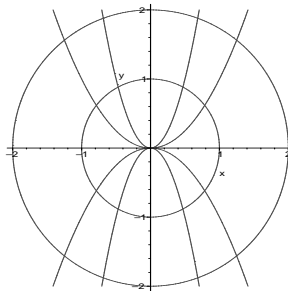


Figure 5: Four parabolas in degenerate configuration. They intersect with each other at a single point, the origin.

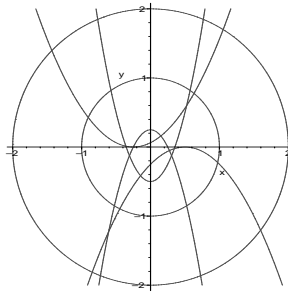


Figure 6: Four parabolas randomly translated.

Consider the problem Π_1 to locate four parabolas

$$\begin{cases} f_1 & : & 2X^2 - Y = 0, \\ f_2 & : & X^2 - Y = 0, \\ f_3 & : & -X^2 - Y = 0, \\ f_4 & : & -2X^2 - Y = 0. \end{cases}$$

They are in degenerate configuration; namely, they intersect with each other at a single point, the origin. (See figure 5.) Suppose a random translation is applied so that they are perturbed to

$$\begin{cases} \tilde{f}_1 & : & 2X^2 - \frac{1}{2} - Y = 0, \\ \tilde{f}_2 & : & X^2 + \frac{1}{2}X + \frac{1}{16} - Y = 0, \\ \tilde{f}_3 & : & -X^2 + X - \frac{1}{4} - Y = 0, \\ \tilde{f}_4 & : & -2X^2 + \frac{1}{4} - Y = 0. \end{cases}$$

(Note, this is not actually a very random perturbation, but it illustrates the idea well.) Then, no three or four of the perturbed parabolas intersect simultaneously at a single point (see figure 6). They are no longer in degenerate configurations. Thus, this random translation is valid.

Now, consider another problem Π_2 to locate four parabolas f_1, \dots, f_4 and the circle

$$g_1 : X^2 + Y^2 - 1 = 0.$$

If we travel on g_1 clockwise from the point $(0, 1)$ to the point $(0, -1)$, then we meet those parabolas in order of f_1, f_2, f_3, f_4 . On the other hand, we meet the perturbed parabolas in order of $\tilde{f}_2, \tilde{f}_1, \tilde{f}_3, \tilde{f}_4$. Thus, even though this random translation is valid, it fails to maintain the designer's intent.

If we consider the problem Π_3 to locate four parabolas f_1, \dots, f_4 and the circle

$$g_2 : X^2 + Y^2 - 4 = 0$$

then the random translation removed degeneracies and still keeps the order of intersections between parabolas and the circle g_2 . Thus, this random translation is valid and also maintains the designer’s intent.

In general, it is neither trivial nor easy to maintain the designer’s intent with application of random translations. The difference between problems Π_2 and Π_3 has to do with the size of the perturbation relative to the size of the circle being tested (i.e. the larger circle was equivalent to having a smaller perturbation). For a given circle size, the perturbation amount could be reduced to be small enough to guarantee correct ordering of the intersections. Similarly, a given numerical perturbation can always be made small enough (since we use exact computation) to be equivalent to any symbolic perturbation. Determining that amount ahead of time, though, is in general a difficult problem.

5.2 Expansion/Contraction for CSG Trees

We have also applied the exact numerical perturbation to resolve degeneracies in a boundary evaluation procedure. The results of this work were presented in a previous short paper [22], so we will not repeat the details here. Instead, we will briefly summarize the main results.

5.2.1 Overview of Procedure

In real-world inputs, degenerate configurations are common. Although a system for exact boundary evaluation, ESOLID, has been developed [19], it assumes general position, and fails on most degenerate inputs. We have implemented an expansion/contraction exact numerical perturbation on top of ESOLID, in order to eliminate problems encountered with degeneracies.

We detect when degeneracies are encountered during boundary evaluation, and apply our perturbation scheme at that time. Since the degeneracy may be encountered high up in a CSG-tree, we need to perturb the objects at the point of the degeneracy. In order to better capture designer’s intent, there is a scheme that decides which object(s) to perturb inward and which outward. Furthermore, we have shown that inward/outward perturbations can

be “pushed down” the CSG-tree to become a set of expansion/contraction operations at the leaves. After perturbation, the following holds:

- Proposition 5.1** 1. *A surface of a primitive solid is expanded/contracted so that it is parallel to that of the original.*
2. *A surface of any solid represented as some subtree rooted at some internal node of a CSG-tree is expanded/contracted so that either 1) it is parallel to that of the original or 2) there is no counterpart in the original.*
3. *An edge of a primitive solid is expanded/contracted so that it is parallel to that of the original.*
4. *An edge of any solid represented as some subtree rooted at some internal node of a CSG-tree is expanded/contracted so that either 1) it is parallel to that of the original or 2) there is no counterpart in the original.*
5. *The centroid of any primitive solid does not move.*

5.2.2 Real-World Examples

The perturbation scheme was applied to a set of real-world models developed under the BRL-CAD solid modeler [4]. ESOLID converts parts from CSG format to an exact B-rep format, and the perturbation is applied as necessary.

We examined three parts from a model of a Bradley Fighting Vehicle, each of which contained degeneracies, and thus could not be evaluated by ESOLID alone. In summary, on these real-world cases, the exact numerical perturbation did indeed allow us to compute the result while maintaining a reasonable designer’s intent. The perturbed versions ran slower than the unperturbed versions (on the portions that both could compute), due to higher bit-length. Filters failed slightly more often. For two examples the increase in time was very modest (40% increase in time), however, one model took significantly more time (7.3 times as long) to compute. In part, this exceedingly longer time was found to be due to secondary effects of the perturbation. Specifically, the perturbation created a situation (having to separate two very close curves that were once only one curve) that our particular evaluation algorithm was not adept at handling. A different boundary evaluation algorithm, however, might have no such difficulty, and thus not require such a significantly longer amount of time.

6 Conclusion

We have presented a formal description of the exact numerical perturbation approach for removing degeneracies. We have described why the approach works, and how it can be implemented in practice.

We have discussed the nature of degeneracies, particularly in solid modeling applications. We have reviewed alternative approaches for handling degeneracies and compared our exact numerical perturbation technique and them to highlight the benefits and drawbacks of our approach. We have shown examples of exact numerical perturbation applied to resolve degeneracies appearing in solid modeling applications.

6.1 Future Work

Among our directions for future work are the following:

We would like to further develop our implementation of the exact numerical perturbation scheme, and test it on a greater range of applications. Due to the complexity of building an exact solid modeling base system, though, we will probably continue to base our work around the ESOLID [19] system.

As mentioned earlier, there are a few cases where the current implementation of exact numerical perturbation does not resolve degeneracies (due to the choice of direction). There are also some situations where multiple degeneracies can cause conflicting information in perturbation directions. Trying to develop a more generalized and systematic approach to specifying perturbation direction would be beneficial for not just exact numerical perturbation, but any perturbation.

Acknowledgements

This work was funded in part by NSF/DARPA CARGO award DMS-0138446 and NSF ITR Award CCR-0220047. The Bradley Fighting Vehicle was provided courtesy of the Army Research Lab.

References

- [1] M. O. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic: A detailed implementa-

- tion. *Computer Aided Design*, 26(6):403 – 416, 1994.
- [2] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Exact efficient computational geometry made easy. In *Proc. of 15th Annual Symposium on Computational Geometry*, pages 341 – 350. ACM, 1999.
 - [3] T. Dubé and C. Yap. The exact computation paradigm. In D. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, Lecture Notes on Computing, pages 452 – 492. World Scientific, 2nd edition, 1995.
 - [4] P. C. Dykstra and M. J. Muuss. The BRL-CAD package an overview. Technical report, Advanced Computer Systems Team, Ballistics Research Laboratory, Aberdeen Proving Ground, MD, 1989.
 - [5] H. Edelsbrunner and E. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66 – 104, 1990.
 - [6] I. Z. Emiris and J. F. Canny. A general approach to removing degeneracies. In *Proc. of 32nd IEEE Symposium on the Foundations of Computer Science*, pages 405 – 413. IEEE, 1991.
 - [7] I. Z. Emiris, J. F. Canny, and R. Seidel. Efficient perturbations for handling geometric degeneracies. *Algorithmica*, 19(1/2):219 – 242, 1997.
 - [8] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of cgal a computational geometry algorithms library. *Software – Practice & Experience*, 30(11):1167 – 1202, 2000.
 - [9] R. T. Farouki, C. A. Neff, and M. A. O’Connor. Automatic parsing of degenerate quadric-surface intersections. *ACM Transactions on Graphics*, 8(3):174 – 208, 1993.
 - [10] S. Fortune. Polyhedral modeling with multiprecision integer arithmetic. *Computer-Aided Design*, 29(2):123 – 133, 1997.
 - [11] S. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. In *Proc. of 14th Annual Symposium on Computational Geometry*, pages 116 – 125. ACM, 1998.

- [12] N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. of 17th Annual Symposium on Computational Geometry*, pages 264 – 273. ACM, 2001.
- [13] M. T. Goodrich, L. J. Guibas, J. Hershberger, and P. J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. of 13th Annual Symposium on Computational Geometry*, pages 284 – 293. ACM, 1997.
- [14] D. Halperin and C. R. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Computational Geometry*, 10(4):273 – 287, 1998.
- [15] J. D. Hobby. Practical segment intersection with finite precision output. *Computational Geometry*, 13(4):199 – 214, 1999.
- [16] C. M. Hoffman. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31 – 41, 1989.
- [17] C. M. Hoffman, J. E. Hopcroft, and M. S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50 – 59, 1989.
- [18] V. Karamcheti, C. Li, and C. Yap. A Core library for robust numerical and geometric computation. In *Proc. of 15th Annual Symposium on Computational Geometry*, pages 351 – 359. ACM, 1999.
- [19] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha. ES-OLID - a system for exact boundary evaluation. *Computer-Aided Design*, 36(2):175 – 193, 2004.
- [20] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer-Aided Design*, 32(11):649 – 662, 2000.
- [21] S. Mehlhorn and M. Näher. *LEDA - A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
- [22] K. Ouchi and J. Keyser. Handling degeneracies in exact boundary evaluation. In *Proc. of 9th ACM Symposium on Solid Modeling and Applications*, pages 321 – 326. ACM, 2004.

- [23] F. P. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.
- [24] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. of 15th Annual Symposium on Computational Geometry*, pages 163 – 172. ACM, 1999.
- [25] R. Seidel. The nature and meaning of perturbations in geometric computing. In *Proc. of 11th Annual Symposium on Theoretical Aspects of Computer Science '94*, LNCS 775, pages 3 – 17. Springer, 1994.
- [26] X. Song, T. W. Sederberg, J. Zheng, R. T. Farouki, and J. Hass. Linear perturbation methods for topologically consistent representations of free-form surface intersections. *Computer Aided Geometric Design*, 21(3):303 – 319, 2004.
- [27] K. Sugihara. A robust and consistent algorithm for intersecting convex polyhedra. In M. Dæhlen and L. Kjeldahl, editors, *Proc. of EURO-GRAPHICS '94*, Computer Graphics Forum, Vol. 13, No. 3, pages C-45 – C-54. Blackwell Association, 1994.
- [28] K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380 – 393, 1989.
- [29] C. Yap. Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computation*, 10(3/4):349 – 370, 1990.
- [30] J. Yu. *Exact Arithmetic Solid Modeling*. Ph.D. thesis, Department of Computer Science, Purdue University, West Lafayette, IN, 1991.