Algorithmica (2000) 27: 5–20
DOI: 10.1007/s004530010002

Algorithmica


# Topology-Oriented Implementation—An Approach to Robust Geometric Algorithms

K. Sugihara,[1] M. Iri,[2] H. Inagaki,[3] and T. Imai[4]

**Abstract.** This paper presents an approach, called the "topology-oriented approach," to numerically robust geometric algorithms. In this approach, the basic part of the algorithm is described in terms of combinatorial and topological computation primarily; this description guarantees robustness of the algorithm because combinatorial and topological computation is never contaminated with numerical errors. However, this part of the algorithm is usually nondeterministic, the flow of processing containing many alternative branches. Hence, numerical computation is used in order to choose the branch that seems the most promising to lead to the correct answer. The algorithm designed in this way is robust and simple. The basic idea of this approach as well as the basic properties of the resulting algorithms is shown with examples.

**Key Words.** Clipping a convex polyhedron, Line-segment Voronoi diagram, Robust implementation, Topological consistency.

**1. Introduction.** There is a great gap between *theoretically correct* geometric algorithms and *practically valid* computer programs. This is mainly because actual computation contains numerical errors; these errors sometimes generate inconsistencies in topology and thus make computer programs fail.

To overcome this difficulty many approaches have been proposed. Roughly speaking, they can be classified into three categories according to how much they rely on the numerical computation.

The approaches in the first category rely on the numerical computation "moderately." They use inexact arithmetic, such as floating-point arithmetic, but they assume that the amount of error in the computation is bounded. On the basis of evaluation of the errors, the predicates computed in the algorithm are divided into reliable and unreliable. The reliable predicates are used positively while the unreliable ones are used carefully in order to avoid inconsistency. This category includes the hidden-variable approach [25], the $\varepsilon$-geometry approach [11], the approximate predicate approach [7], [8], the tolerance approach [33], [45], and other error-analysis approaches [5], [14], [15], [30]. In these approaches, individual problems more or less require their own sophistications, but once they are implemented, they run fast because the floating-point arithmetic can be executed quickly.

[1] Department of Mathematical Engineering and Information Physics, Graduate School of Engineering, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan.
[2] Department of Information and System Engineering, Faculty of Science and Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan.
[3] Department of Information and Computer Engineering, Toyota National College of Technology, 2-1 Eiseicho, Toyota-shi, Aichi 471-8525, Japan.
[4] Department of Design and Information Sciences, Wakayama University, 930 Sakaedani, Wakayama 640-8510, Japan.

Received December 12, 1996; revised June 1, 1998. Communicated by S. Fortune.

The approaches in the second category rely on the numerical computation "completely." They use exact arithmetic such as integer arithmetic and thus always obtain correct predicates. In the early days their primal interest was how to achieve exact arithmetic [10], [28], [40], [44], and how to cope with degeneracy [6], [43]. It is now considered a standard method to use multiple-precision arithmetic together with symbolic perturbation. This method seems promising because theoretical algorithms can be implemented directly. The main practical issue is how to decrease the cost of exact computation.

A typical technique is the floating-point filter, in which the predicates are first computed in floating-point arithmetic, and only when it turns out to be unreliable is exact arithmetic used [1], [9], [21], [38]. A more sophisticated version is an adaptive filter in which the precision of the filter is adjusted [31]. Methods to decrease the required precision are also proposed. They include the basis-reduction technique [3], [4], the modular arithmetic techniques [2], [17], and the implicit representation technique [23]. Because of these acceleration techniques, many geometric algorithms concerned with low-degree objects such as lines and planes can be implemented into practically fast software [24].

The approaches in the third category rely on numerical values the least. In other words, they aim at algorithms that are robust even if numerical errors are large. This category includes the axiomatic approach [22], [29], in which numerical computation is done only when it is not redundant; this approach is also called the parsimonious approach.

In this paper we present another approach to robustness belonging to the third category, which we call the "topology-oriented approach" or the "combinatorial-abstraction approach." In this approach the basic part of an algorithm is described in terms of combinatorial and topological computation primarily, and numerical computation is used as secondary information. A remarkable point in this approach is that the inconsistency issue is completely separated from the numerical error issue. We need not consider the amount of numerical errors when we construct the basic part of the robust algorithm; in this sense the design process is simple.

We proposed the first idea of this approach in 1988 [39]. Since then we have been developing this approach by applying the idea to many geometric problems, such as construction of various Voronoi/Delaunay diagrams in two and three dimensions [18]–[20], [27], [35], [41], [42], construction of three-dimensional convex hulls [26], [36], and intersection of convex polyhedra [37].

In what follows, we summarize the basic idea behind these works from a unifying point of view, and thus try to clarify the general idea of the topology-oriented approach.

## 2. Robustness against Numerical Errors and Consistency in Topology.

Let $P$ be a geometric problem, and let $f$ be a theoretical algorithm to solve $P$. By a "theoretical" algorithm, we mean an algorithm that is designed assuming precise arithmetic, namely, one whose correctness is based on the assumption that no numerical error takes place in the computation.

The algorithm $f$ can be considered a mapping from the set $\Xi(P)$ of all possible inputs to the set $\Omega(P)$ of all possible outputs. Each input $X \in \Xi(P)$ represents an instance of the problem $P$, and the corresponding output $f(X) \in \Omega(P)$ is a solution of the problem instance.

Both the input and the output can be divided into the "combinatorial and/or topological part" ("topological part" for short) and the "metric part." We represent the topological part by a subscript T and the metric part by a subscript M. More specifically, the input $X$ is divided into the topological part $X_{\mathrm{T}}$ and the metric part $X_{\mathrm{M}}$, and the output $f(X)$ is divided into the topological part $f_{\mathrm{T}}(X)$ and the metric part $f_{\mathrm{M}}(X)$.

For example, suppose that $P$ is the problem of constructing the Voronoi diagram for a finite number of given points in the plane. Then the topological part $X_{\mathrm{T}}$ of the input consists of a single integer to represent the number $n$ of points, and the metric part $X_{\mathrm{M}}$ is the set of the $n$ pairs of coordinates of the points: $X_{\mathrm{T}} = \{n\}$ and $X_{\mathrm{M}} = \{x_1, y_1, \ldots, x_n, y_n\}$. The topological part $f_{\mathrm{T}}(X)$ of the output is the planar graph structure consisting of the Voronoi vertices and the Voronoi edges, and the metric part $f_{\mathrm{M}}(X)$ consists of the coordinates of the Voronoi vertices and the directions of the infinite Voronoi edges.

For another example, suppose that $P$ is the problem of constructing the intersection of two convex polyhedra in three-dimensional space. Then $X_{\mathrm{T}}$ consists of the incidence structure among the vertices, the edges, and the faces of the two polyhedra, and $X_{\mathrm{M}}$ consists of the three-dimensional coordinates of the vertices and/or the coefficients of the face equations. Similarly, $f_{\mathrm{T}}(X)$ and $f_{\mathrm{M}}(X)$ are the topological part and the metric part, respectively, of the polyhedron which is the intersection of the two input polyhedra.

Let $\tilde{f}$ denote an actually implemented computer program to solve $P$. The program $\tilde{f}$ may be a simple translation of the algorithm $f$ into a programming language, or it may be something more sophisticated aiming at robustness. The program $\tilde{f}$ can also be considered a mapping from the input set to the output set. However, in actual situations, the program runs in finite-precision arithmetic, and consequently the behavior of $\tilde{f}$ is usually different from that of $f$.

The program $\tilde{f}$ is said to be *numerically robust* (or *robust* for short) if $\tilde{f}(X)$ is defined for any input $X$ in $\Xi(P)$. In other words, $\tilde{f}$ is robust if it defines a total (not partial) function from $\Xi(P)$ to superset $\hat{\Omega}(P)$ of $\Omega(P)$, i.e., if the program always carries out the task, ending up with some output, neither entering into an endless loop nor terminating abnormally.

The program $\tilde{f}$ is said to be *topologically consistent* (or *consistent* for short) if $\tilde{f}$ is robust and $\tilde{f}_{\mathrm{T}}(X) \in \Omega_{\mathrm{T}}(P)$ for any $X \in \Xi(P)$. In other words, $\tilde{f}$ is consistent if the topological part $\tilde{f}_{\mathrm{T}}(X)$ of the output coincides with the topological part $f_{\mathrm{T}}(X')$ of the correct solution of some instance $X'$ (not necessarily equal to $X$) of the problem $P$.

Our goal is to construct $\tilde{f}$ that is at least robust and hopefully consistent.

**3. Basic Idea.** In the topology-oriented approach, we start with the following assumptions.

ASSUMPTION 1. Logical and combinatorial computations can be done correctly, whereas numerical computations contain errors.

ASSUMPTION 2. There is no a priori bound available on the amount of numerical errors.

The readers might feel that Assumption 2 is too pessimistic, because a certain precision is usually guaranteed in actual computation. However, we do not like to be pessimistic, but just want to show that, even if we do not rely on numerical results at all, we can still design robust algorithms. As we show later, Assumption 2 necessarily separates the robustness issue from the error-analysis issue, and thus makes the design of the algorithm simpler.

Suppose that we are given a geometric problem $P$ together with a conventional algorithm $f$ to solve $P$. We construct a robust implementation $\tilde{f}$ of $f$ in the following three steps.

*Step I.* Collect purely topological properties that should be satisfied by the solutions of the problem $P$ and that can be checked efficiently. Let $Q$ be the set of such properties.

By "purely topological properties" we mean those properties that can be represented by only combinatorial and/or topological terms, without referring to numerical values. On the other hand "can be checked efficiently" means that the computational cost for checking the property is acceptable. For example, suppose that the algorithm $f$ runs in $O(n \log n)$ time, and $q$ is a topological property possessed by any solution of the problem $P$. If it is NP-hard to check $q$, we should definitely not put $q$ in $Q$. In general, whether we can put $q$ in $Q$ or not depends on the time allowed in applications.

EXAMPLE 1 (Clipping a Convex Polyhedron by a Plane). Suppose that we are given a convex polyhedron $\Pi$ and a half-space $H$, and we want to construct the intersection $\Pi \cap H$. Let $\partial H$ denote the boundary plane of $H$. Constructing $\Pi \cap H$ is equivalent to cutting $\Pi$ by the plane $\partial H$ and removing one part, as shown in Figure 1(a).

The set $V$ of vertices and the set $E$ of edges of $\Pi$ form a graph $G = (V, E)$. We call this graph the *vertex–edge graph* of $\Pi$. For any subset $V' \subseteq V$, let $G(V')$ denote the subgraph of $G$ induced by $V'$. The cutting plane $\partial H$ divides the vertex set $V$ into two subsets, say $V_1$ and $V_2$: $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$.

Now we can see the following properties:

(P1.1) The vertex–edge graph $G = (V, E)$ of $\Pi$ is planar.
(P1.2) Both $G(V_1)$ and $G(V_2)$ are connected.

Property (P1.1) holds because the boundary of a convex polyhedron is homeomorphic to a sphere. Property (P1.2) comes from the fact that $\Pi$ is convex and $\partial H$ is a plane. Note that these properties come from the convexity of $\Pi$ and the convexity is a metric property. However, both of the resulting properties are stated in combinatorial terms only. Hence they are purely topological properties.

When we cut $\Pi$ by $\partial$H, the new face $\Pi \cap \partial H$ is generated. The next property also holds.

(P1.3) The new face $\Pi \cap \partial H$ is a convex polygon.

However, this property is not purely topological, because whether a face is convex or not depends on the coordinates (i.e., numerical values) of the vertices of the face.

Thus, (P1.1) and (P1.2) belong to $Q$, but (P1.3) does not.

*Step II.* Describe the basic part of the algorithm only in terms of combinatorial and topological computation in such a way that the properties in $Q$ are guaranteed. Here we need not consider degenerate cases.

Combinatorial and topological computations can always be done correctly, and hence we can design this part of the algorithm without worrying about numerical errors. We call this part of the algorithm the *topological skeleton*.

Note that numerical computation is not assumed to be exact, and hence we cannot detect whether the input is degenerate. This is the reason why we ignore degenerate cases. The consequence of this are discussed in Section 5.

The topological skeleton designed in Step II does not specify the behavior of the algorithm uniquely. It usually contains nondeterministic branches.

EXAMPLE 1 (*continued*). On the basis of properties (P1.1) and (P1.2), we can construct the topological skeleton of the algorithm in the following way. The statements in brackets are comments describing what we actually want to do, and the statements in parentheses refer to the example shown in Figure 1.

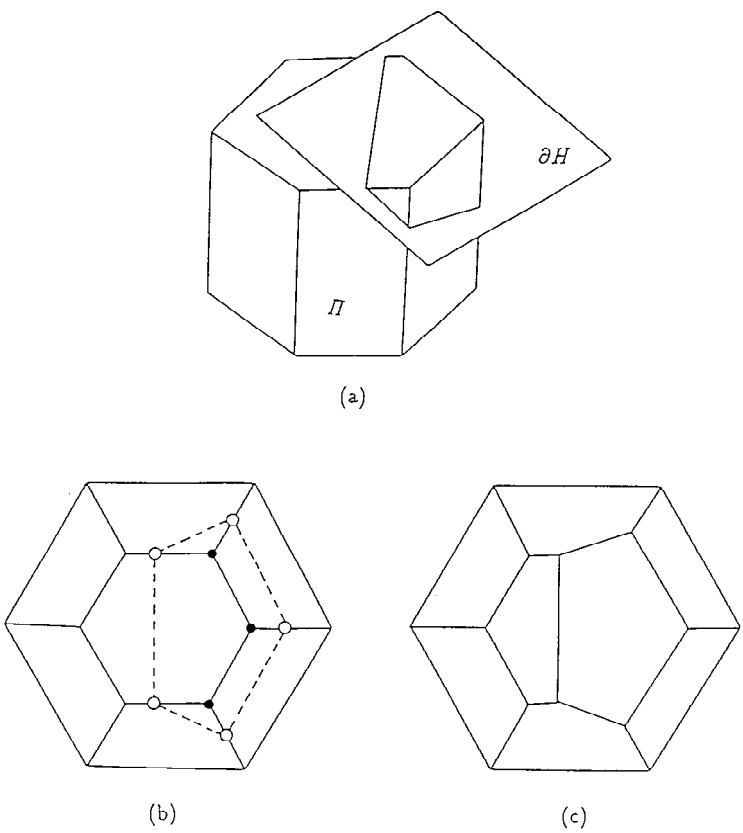


**Fig. 1.** Topological skeleton of the intersection operations.

ALGORITHM 1 (Topological Skeleton for Intersection $\Pi \cap H$).

*Input*: Planar graph $G = (V, E)$ [$G$ is supposed to be the vertex–edge graph of $\Pi$].
*Output*: Planar graph $G' = (V', E')$ [$G'$ is hopefully the vertex–edge graph of $\Pi \cap H$].
*Procedure*:

1. Divide $V$ into $V_1$ and $V_2$ in such a way that (P1.2) is satisfied [hopefully the vertices in $V_1$ are outside $H$ and those in $V_2$ are inside $H$] (in Figure 1(b), the vertices in $V_1$ are represented by solid circles).
2. On each edge connecting $V_1$ and $V_2$, generate a new vertex (the vertices represented by open circles in Figure 1(b)).
3. Generate a new circuit passing through all the new vertices and separating $V_1$ from $V_2$ (the circuit represented by broken lines in Figure 1(b)).
4. Remove the vertices in $V_1$ and the edges incident to them; name the resulting graph $V'$, and report it (the graph in Figure 1(c)).

Note that in this algorithm numerical values are not referred to at all. Steps 2–4 are deterministic. Only Step 1 is nondeterministic; there are in general many possible ways to divide $V$ into $V_1$ and $V_2$. However, as far as (P1.2) is fulfilled, all the steps can be done consistently and the resulting graph $G'$ is planar.

As shown in this example, the topological skeleton usually contains nondeterministic branches. Hence in general the flow of processing can be represented by a rooted acyclic directed graph as shown in Figure 2. The algorithm starts at the root node at the top, and goes downward, choosing one of the branches nondeterministically at each node of the graph; the algorithm terminates when it reaches one of the leaf nodes. This graph contains the path reaching the correct solution of $P$ as shown by the bold path in the figure, but at

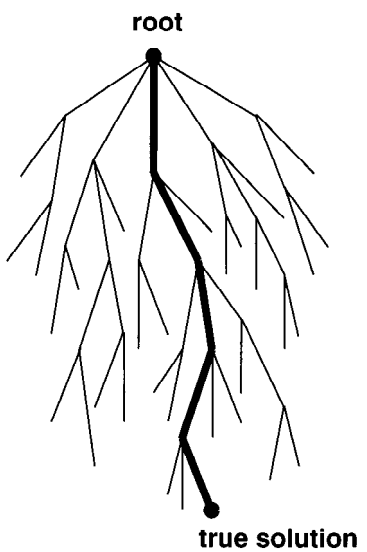


**Fig. 2.** Nondeterministic branches in the topological skeleton of a geometric algorithm.

this moment we do not know which is the correct path because the topological skeleton is described only by combinatorial and topological computations. It should be noted, however, that in this graph any path from the root node represents a consistent behavior of the algorithm in the sense that the topological properties in $Q$ are preserved at all nodes.

*Step III*. Conduct numerical computations at each node of the tree in order to choose the branch that is most likely to lead to the correct solution of the problem $P$.

Step III adds numerical information to the topological skeleton, and thus makes the behavior of the algorithm deterministic. We denote the resulting implementation of the algorithm by $\tilde{f}$.

EXAMPLE 1 (*continued*). We use the results of numerical computation in order to make Step 1 in Algorithm 1 deterministic. More specifically, we add vertex $v \in V$ to $V_1$ if the numerical computation tells us that $v$ is outside $H$ and the addition of $v$ to $V_1$ does not violate (P1.2).

Figure 3 shows an example of the behavior of the computer program constructed in this way. Figure 3(a) is the output of the program which cuts a large cube by many planes tangent to a common paraboloid of revolution (there is no special reason in choosing this surface; any convex smooth surface can be used similarly). All the floating-point computations were done in single precision. The input is highly degenerated in the sense that many cutting planes have a common point of intersection. Figure 3(b) is the figure around the degenerate vertex at the top (point $q$ in Figure 3(a)) magnified by

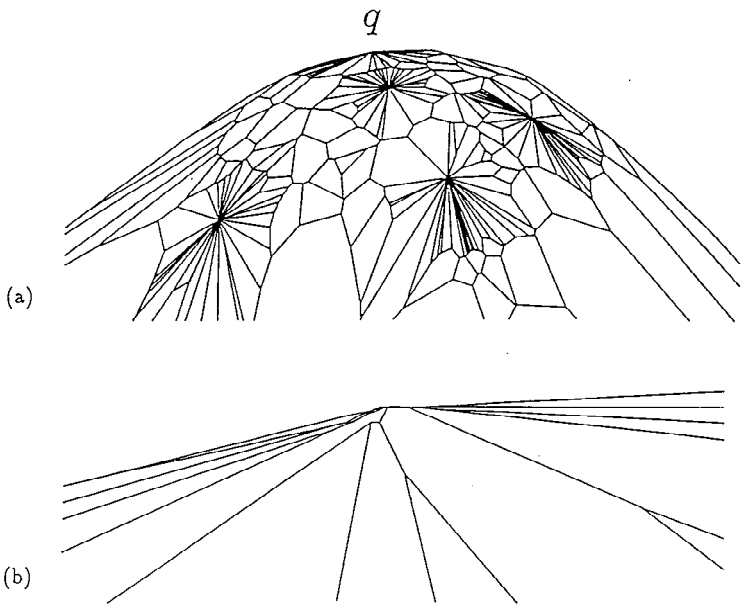


**Fig. 3.** Output of Algorithm 1 for degenerate input: (a) output, (b)!diagram around the degenerate vertex at the top magnified by $5 \times 10^5$.

$5 \times 10^5$. We can see that there is a complicated microstructure around such a degenerate point. However, in this example the disturbance arises only in very small areas, and we cannot see it in the normal scale as shown in Figure 3(a); hence we can use this program in applications such as graphics where strictly exact solutions are not necessarily required.

Implementation $\tilde{f}$ of the algorithm based on Steps I–III behaves in the following way. If the arithmetic is precise, the correct path is chosen and the correct solution is obtained as the output of $\tilde{f}$. If the arithmetic is not precise, on the other hand, the output of $\tilde{f}$ may not be correct, but at least it satisfies the topological properties in $Q$ and hopefully it can be considered an approximation of the correct solution. Thus, through Steps I–III we can construct a program $\tilde{f}$ that is at least robust.

It should also be noted that we need not be bothered by degenerate inputs. We assume that numerical errors cannot be avoided, and consequently we cannot judge whether the input is degenerate. Even if the numerical computation tells us that degeneracy takes place, we need not believe it. It simply implies that the situation is *close to* degeneracy. It might sound paradoxical, but once we admit the existence of numerical errors, we need not consider degeneracy and consequently the implementation of an algorithm becomes much simpler.

**4. Another Example.** The topology-oriented approach shows its merit clearly when it is applied to "high-degree' problems, where exact arithmetic is very expensive (see [23] for the "degree" of a geometric problem). For an example of such problems, here we consider the Voronoi diagram for line segments.

EXAMPLE 2 (Voronoi Diagram Generated by Line Segments). Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of mutually disjoint closed line segments in the plane $\mathbf{R}^2$. For any point $p \in \mathbf{R}^2$ we define $d(p, s_i) = \inf_{q \in s_i} d(p, q)$. We call the set

$$R(s_i) = \{p \in \mathbf{R}^2 \mid d(p, s_i) < d(p, s_j) \text{ for any } j \neq i\}$$

the *Voronoi region* of $s_i$. The partition of the plane into $R(s_1)$, $R(s_2)$, $\ldots$, $R(s_n)$ and their boundaries is called the *Voronoi diagram* for $S$, and is denoted by $\mathrm{Vor}(S)$.

Let $p_{2i-1}$ and $p_{2i}$ be the endpoints of $s_i$, and let $\Pi$ be the set of all the endpoints of line segments in $S$, and let $S^{\mathrm{o}} = \{s_1^{\mathrm{o}}, s_2^{\mathrm{o}}, \ldots, s_n^{\mathrm{o}}\}$ be the set of open line segments that are obtained when we remove the endpoints from the line segments in $S$. To construct $\mathrm{Vor}(S)$, we first construct $\mathrm{Vor}(\Pi)$ by the topology-oriented algorithm for the ordinary Voronoi diagram presented in [41] and [42], and next modify it step by step by adding elements of $S^{\mathrm{o}}$.

During the modification we treat open line segments and their endpoints as distinct generators. Therefore, the Voronoi region of $s_i^{\mathrm{o}}$ and that of $p_{2i-1}$ (resp. $p_{2i}$) share a common boundary edge; we assume that this edge is on the line passing through $p_{2i-1}$ (resp. $p_{2i}$) perpendicular to $s_i^{\mathrm{o}}$. Those virtual edges will be removed at the end of the construction.

Suppose that we have constructed the Voronoi diagram $\mathrm{Vor}(\Pi \cup \{s_1^{\mathrm{o}}, s_2^{\mathrm{o}}, \ldots, s_{i-1}^{\mathrm{o}}\})$ and want to add $s_i^{\mathrm{o}}$. Let $V$ and $E$ be the set of vertices and edges of $\mathrm{Vor}(\Pi \cup \{s_1^{\mathrm{o}}, s_2^{\mathrm{o}}, \ldots, s_{i-1}^{\mathrm{o}}\})$.

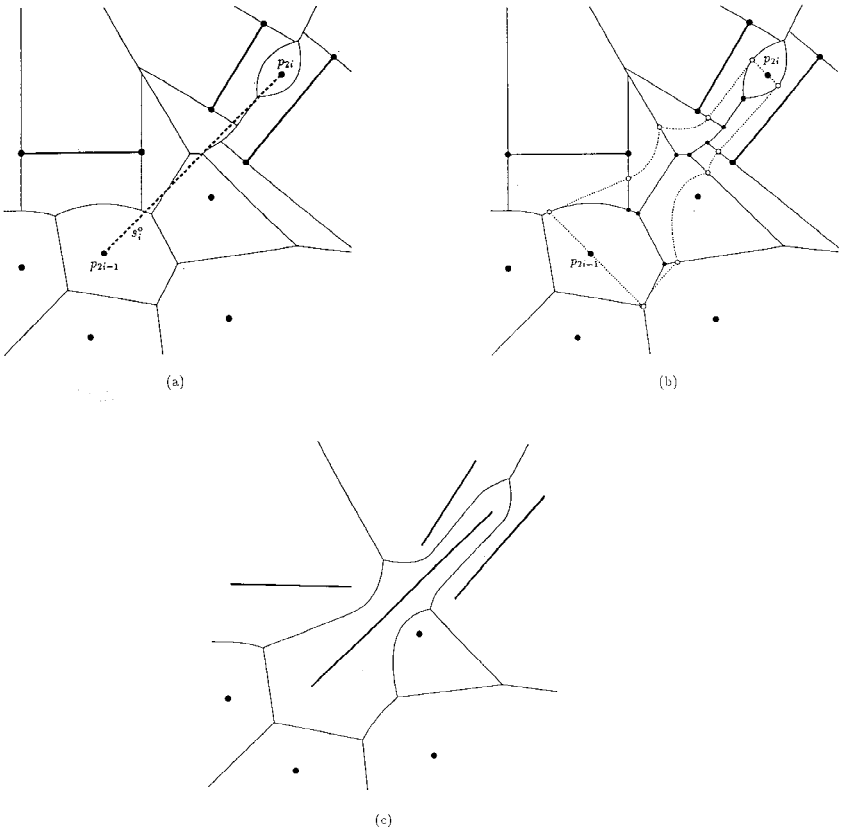


**Fig. 4.** Topological skeleton of the addition of a new open line segment.

Then $G = (V, E)$ can be considered a planar graph embedded in the plane. An example of changing the diagram from $\mathrm{Vor}(\Pi \cup \{s_1^o, s_2^o, \ldots, s_{i-1}^o\})$ to $\mathrm{Vor}(\Pi \cup \{s_1^o, s_2^o, \ldots, s_i^o\})$ is depicted in Figure 4, where $\mathrm{Vor}(\Pi \cup \{s_1^o, s_2^o, \ldots, s_{i-1}^o\})$ is shown by the solid lines in (a) and the new line segment $s_i^o$ is shown by the broken line. We first find the set $V'$ of vertices and the set $E'$ of edges that should be entirely included in the Voronoi region of $s_i^o$; elements of $V'$ are represented by small solid circles in Figure 4(b). Sometimes there exist edges that connect two vertices in $V'$ but are not included in $E'$. These edges are partially included in the Voronoi region of $s_i^o$, that is, both parts of these edges close to the endpoints are included in the Voronoi region of $s_i^o$, whereas the middle parts are not included. We denote the set of those edges by $E''$. On the other hand, there is no edge whose middle part is included in the Voronoi region of $s_i^o$ but whose endpoints are not included; this is because we add the endpoints first and the open line segments next. We next generate new vertices on the edges, one new vertex on each edge in $E'$ and two new vertices on each edge in $E''$, as shown by the open circles in Figure 4(b). Then we connect these new vertices by new edges in such a way that they form a circuit separating $V'$ from $V - V'$, as shown by the broken edges in

Figure 4(b). Finally we remove the vertices in $V'$ and the edges incident to them as shown in Figure 4(c).

In this process, we can observe the following topological properties:

(P2.1) The subgraph $(V', E')$ is a tree (i.e., a connected graph without circuit).
(P2.2) The subgraph $(V', E')$ contains a path connecting $R(p_{2i-1})$ and $R(p_{2i})$.

The graph $(V', E')$ is connected because the Voronoi region $R(s_i)$ is connected, and $(V', E')$ does not contain a circuit because none of the old Voronoi regions disappears entirely when $s_i^o$ is added. Therefore, property (P2.1) holds. Property (P2.2) holds because the new line segment $s_i^o$ connects $p_{2i-1}$ and $p_{2i}$. On the basis of these observations, we can construct the topological skeleton of the above process in the following way.

ALGORITHM 2 (Topological Skeleton of the Voronoi Diagram for Line Segments).

*Input*: Planar graph $G = (V, E)$ [$G$ is supposed to be the graph of Vor($\Pi \cup \{s_1^o, s_2^o, \ldots, s_{i-1}^o\}$)]
*Output*: Planar graph $G^* = (V^*, E^*)$ [hopefully $G^*$ is the graph of Vor($\Pi \cup \{s_1^o, s_2^o, \ldots, s_i^o\}$)].
*Procedure*:

1. Select subsets $V' \subseteq V$ and $E' \subseteq E$ that satisfy (P2.1) and (P2.2) [$V'$ and $E'$ are supposed to be the set of vertices and that of edges that should be deleted completely in the addition of $s_i^o$]. Let $E''$ be the edges that connect two vertices in $V'$ but are not included in $E'$.
2. Generate a new vertex on each edge in $E'$ and generate two new vertices on each edge in $E''$.
3. Generate a new circuit passing through all the new vertices and separating $V'$ from $V - V'$.
4. Remove the vertices in $V'$ and the edges incident to them. Name the resulting graph $G^* = (V^*, E^*)$ and report it.

In this algorithm, Step 1 is nondeterministic. We use the result of numerical computation in order to choose as $V'$ the set of vertices that are most likely to be deleted in the addition of $s_i^o$. Thus, we can make Algorithm 2 deterministic.

Figure 5 shows examples of the output of the computer program based on Algorithm 2. The input set of line segments in Figure 5(a) was obtained in such a way that they were generated one by one at random and if they intersect the newer one was cut near the point of intersection. On the other hand, the input set of line segments in Figure 5(b) was obtained in such a way that first they were generated at random and next pairs of mutually intersecting line segments were flipped until no intersection remains.

This method was extended to the construction of the Voronoi diagram for polygons [16].

There are many other examples of topology-oriented implementations of geometric algorithms. They include the incremental construction of two-dimensional Voronoi diagrams [41], [42], the divide-and-conquer construction of two-dimensional Voronoi diagrams [27], construction of the three-dimensional convex hull [26], [36], construction of line arrangements in the plane [34], construction of three-dimensional Voronoi

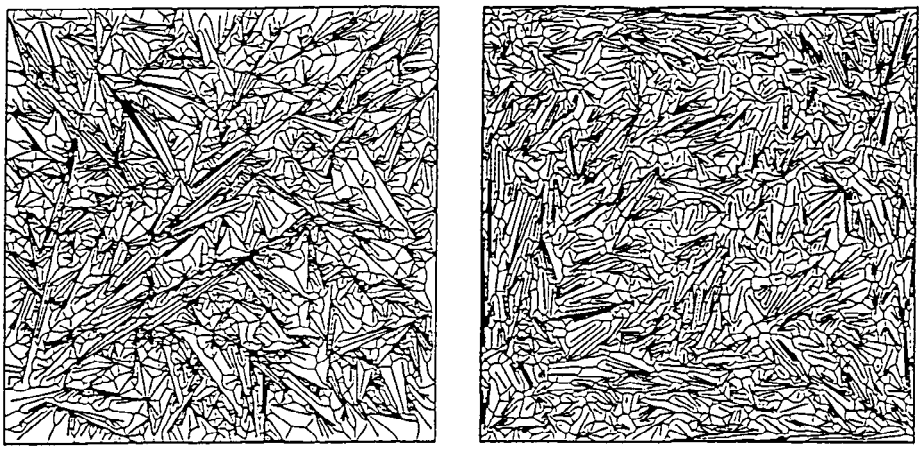


**Fig. 5.** Outputs of Algorithm 2.

diagrams [19], [20], and approximate construction of the Voronoi diagram for general figures in the plane [35].

**5. Discussion.** Here we consider some general properties of the topology-oriented algorithms.

*Robustness.* A topology-oriented algorithm is completely robust in the sense that it does not require any minimum precision in numerical computation. All possible behavior is specified by the topological skeleton, and therefore even if numerical precision is very poor (or even if all the results of numerical computation are replaced by random numbers), the algorithm still carries out the task and generates some output.

*Topological Consistency.* Whether the algorithm is topologically consistent depends on the chosen set $Q$ of purely topological properties. The topology-oriented implementation guarantees that the output satisfies all the properties in $Q$. In general, however, $Q$ gives only a necessary condition for the output to belong to the set $\Omega(P)$ of all the possible solutions of the problem $P$; it does not necessarily give a sufficient condition. This is because the purely topological characterization of the solution set is not known for many geometric problems, and even if it is known, it is usually time-consuming to check the conditions (recall that $Q$ contains only those properties that can be checked efficiently).

Hence, topological consistency can be attained for a limited number of problems. A trivial example is the problem of constructing a convex hull in the plane. For this problem, any cyclic sequence of three or more vertices chosen from the input points can be the solution of a perturbed version of the input, so that topological consistency can be easily attained.

More nontrivial examples arise in the class of problems related to convex polyhedra. The topological structures of convex polyhedra can be characterized by Steinitz's theorem, which says that graph $G$ is a vertex–edge graph of a convex polyhedron if and only if $G$ is a 3-connected planar graph with four or more vertices [32]. Because of this theorem we can see that Algorithm 1 in Example 1 is topologically consistent. Actually we can prove that if the input graph $G$ is a 3-connected planar graph, then the output $G'$ is also a 3-connected planar graph. Hence, the output of Algorithm 1 is the vertex–edge graph of some polyhedron, that is, the output is the vertex–edge graph of the solution of some instance of the problem though it is not necessarily the given instance.

On the other hand, since there is no known necessary and sufficient topological condition for a graph to be a vertex–edge graph of a Voronoi diagram for line segments, we cannot prove that Algorithm 2 in the previous example is topologically consistent or that the chosen set $Q$ of topological properties gives a sufficient condition.

For two-dimensional Voronoi diagram for points, necessary and sufficient conditions are known [12], [13]. However, these conditions require much time to check, and hence cannot be included in $Q$. Hence topological consistency is not easy to achieve, either.

*Convergence.* If the input to the algorithm is not degenerate, the output converges to the correct solution as the computation becomes more and more precise, because the correct branch of the processing is chosen with sufficiently high precision. However, the speed of convergence cannot be stated in a unifying manner, because it depends on the individual problem and on the implementation of numerical computation.

The situation is different for degenerate input. If the algorithm is topologically consistent, the output converges to an infinitesimally perturbed version of the correct solution. In any high precision, the true degenerate output cannot be obtained, because degenerate cases are not taken into account in Step II of the implementation. For example, suppose that the cutting plane $\partial H$ goes through a vertex $v$ of the polyhedron $\Pi$ in Example 1. Then our algorithm classifies the vertex $v$ either inside or outside the half-space $H$. If $v$ is classified inside, the algorithm connects this vertex to a new vertex by a very short edge, which does not disappear in the output data even if the precision increases to infinity, though its length converges to 0.

If the algorithm is not topologically consistent and the input is degenerate, the output also converges to a certain structure, but this structure may not be obtained by any perturbation of the input. This situation reminds us of the way of introducing real numbers from rational numbers. The set of real numbers is obtained by the "completion" procedure such that the limits of any Cauchy sequences are added to the set of rational numbers. It might be thought that the output of the topology-oriented algorithm converges to something obtained by a similar "completion" procedure.

*Time Complexity.* The time complexity of the program $\tilde{f}$ implemented by the topology-oriented approach is either equal to or greater than that of the original algorithm $f$. Let the time complexity of $f$ be $O(g(m))$. There are two factors that may increase the time complexity of $\tilde{f}$.

First, we have to check the properties in $Q$. Let $Q$ consist of properties $q_1, q_2, \ldots, q_k$. Suppose that to check $q_i$ requires $O(t_i(n))$ time, and that $q_i$ is checked $O(u_i(n))$ times in the whole program. Let $h(n) = \max_{1 \le i \le k} t_i(n)u_i(n)$. Then the time complexity of

the program $\tilde{f}$ is $O(g(n) + h(n))$. Hence if $O(h(n))$ is greater than $O(g(n))$, the time complexity increases.

The above argument is much too simplified; actually that is true only when $\tilde{f}$ behaves like $f$. Suppose that the input is far from degenerate and the arithmetic precision is high enough to compute all the predicates correctly. Then $\tilde{f}$ behaves like $f$, with the additional cost for checking the properties in $Q$. Thus the above argument is true.

On the other hand, suppose that the input is relatively close to degenerate (or, equivalently, the precision in the arithmetic is relatively low). Then $\tilde{f}$ may behave quite differently from $f$ because numerical errors often generate complicated microstructures such as shown in Figure 3(b). In that case, the time complexity of the actual program may become larger than that of the theoretical algorithm. This is the second factor that may increase the time complexity. However, this happens only when the precision is too poor to get a meaningful output. Usually the program $\tilde{f}$ behaves like the original algorithm $f$ with the additional cost of checking the topological properties.

**6. Concluding Remarks.** We have presented the basic idea of the topology-oriented approach to numerically robust geometric algorithms, and have given examples of algorithms designed in this approach. Since we can separate the topological-inconsistency issue from the error-analysis issue completely, the algorithm designed in this approach has the following advantages:

(1) No matter how large the numerical errors are that may take place, the algorithm never fails; it always carries out the task and gives some output.
(2) The output is guaranteed to satisfy the topological properties $Q$ used in the topological skeleton of the algorithm.
(3) For a nondegenerate input, the output converges to the correct solution as the precision in computation becomes higher.
(4) The structure of the algorithm is simple because exceptional branches for a degenerate input are not necessary.

We did not discuss the numerical computation. This is mainly because in our approach we can separate the error-analysis issue from the design of the robust algorithm. However, the quality of the output substantially depends on the quality of the numerical computations used in the algorithm. Therefore, once we have constructed a robust algorithm by the topology-oriented approach, we should next tune up the numerical part of the algorithm in order to get a better output. Refer to [42] for an example of tuning up the numerical computation.

There are some limitations in our approach. First, we have to find the set $Q$ of purely topological properties that should be satisfied by the correct output, and next we have to describe the topological skeleton of the algorithm using $Q$. These steps are not trivial. Actually the topology-oriented approach gives a "principle" for designing robust software, and there are freedoms in applying this principle to individual problems; for example, the freedom in choosing the properties and in choosing the ways of numerical computations.

Another limitation is that the output is in general an approximation of the true answer. Hence, this is not appropriate if we need strictly correct answers.

However, in many cases we want just approximations. An example of such cases is computer graphics, where invisible errors are acceptable. Another example is the case where the input itself is an approximation (for example, the application of the Voronoi diagram to dot-pattern analysis for the dots extracted from a digital image). Moreover, sometimes we have to abandon the correct answers because they require too much computational cost. Our approach is suitable in these cases. An example is the Voronoi diagram for line segments (Example 2). Another example is the approximate construction of the Voronoi diagrams for general figures [35].

**Acknowledgments.** The authors express their thanks to the three anonymous referees and Dr. Steve Fortune for their valuable comments and suggestions, which improved this paper very much.

## References

[1] Benouamer, M., Michelucci, D., and Peroche, B., 1993: Error-free boundary evaluation using lazy rational arithmetic—a detailed implementation. *Proceedings of the* 2*nd Symposium on Solid Modeling and Applications*, Montreal, May 1993, pp. 115–126.

[2] Brönnimann, H., Emiris, I. Z., Pan, V. Y., and Pion, S., 1997: Computing exact geometric predicates using modular arithmetic with single precision. *Proceedings of the* 13*th Annual ACM Symposium on Computational Geometry*, Nice, June 1997, pp. 174–182.

[3] Brönnimann, H., and Yvinec, M., 1997: Efficient exact evaluation of signs of determinants. *Proceedings of the* 13*th Annual ACM Symposium on Computational Geometry*, Nice, June 1997, pp. 166–173.

[4] Clarkson, K. L., 1992: Safe and effective determinant evaluation. *Proceedings of the* 33*rd IEEE Symposium on Foundation of Computer Science*, pp. 387–395.

[5] Dobkin, D., and Silver, D., 1988: Recipes for geometry and numerical analysis—Part 1, An empirical study. *Proceedings of the* 4*th Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, June 1988, pp. 93–105.

[6] Edelsbrunner, H., and Mücke, E. P., 1988: Simulation of simplicity—A technique to cope with degenerate cases in geometric algorithms. *Proceedings of the* 4*th Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, June 1988, pp. 118–133.

[7] Fortune, S., 1989: Stable maintenance of point-set triangulations in two dimensions. *Proceedings of the* 30*th IEEE Symposium on Foundations of Computer Science*, Research Triangle Park, October 1989, pp. 494–499.

[8] Fortune, S., 1995: Numerical stability of algorithms for 2D Delaunay triangulations. *International Journal of Computational Geometry and Applications*, vol. 5, pp. 193–213.

[9] Fortune, S., and Van Wyk, C. J., 1996: Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, vol. 15, pp. 223–248.

[10] Greene, D. H., and Yao, F., 1986: Finite-resolution computational geometry. *Proceedings of the* 27*th IEEE Symposium on Foundations of Computer Science*, Toronto, October 1986, pp. 143–152.

[11] Guibas, L., Salesin, D., and Stolfi, J., 1989: Epsilon geometry—building robust algorithms from imprecise calculations. *Proceedings of the* 5*th Annual ACM Symposium on Computational Geometry*, Saarbrücken, June 1989, pp. 208–217.

[12] Hiroshima, T., and Sugihara, K., 1996: Recognition of Delaunay graphs. *Proceedings of the Korea–Japan Joint Workshop on Algorithms and Computation*, Seoul, July 1996, pp. 102–109.

[13] Hodgson, C. D., Rivin, I., and Smith, W. D., 1992: A characterization of convex hyperbolic polyhedra and of convex polyhedra inscribed in the sphere. *Bulletin of the American Mathematical Society*, vol. 27, pp. 246–251.

[14] Hoffmann, C. M., 1989: *Geometric & Solid Modeling*. Morgan Kaufmann, San Mateo.

[15] Hoffmann, C. M., Hopcroft, J., and Karasick, M., 1988: Towards implementing robust geometric computations. *Proceedings of the 4th Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, June 1988, pp. 106–117.

[16] Imai, T., 1996: A topology-oriented algorithm for the Voronoi diagram of polygons. *Proceedings of the 8th Canadian Conference in Computational Geometry*, Ottawa, August 1996, pp. 107–112.

[17] Imai, T., 1996: How to get the sign of integers from their residuals. *Abstracts of the 9th Franco–Japan Days on Combinatorics and Optimization*, p. 7.

[18] Imai, T., and Sugihara, K., 1994: A failure-free algorithm for constructing Voronoi diagrams of line segments (in Japanese). *Transactions of Information Processing Society of Japan*, vol. 35, pp. 1966–1977.

[19] Inagaki, H., and Sugihara, K., 1994: Numerically robust algorithm for constructing constrained Delaunay triangulation. *Proceedings of the 6th Canadian Conference on Computational Geometry*, Saskatoon, August 1994, pp. 171–176.

[20] Inagaki, H., Sugihara, K., and Sugie, N., 1992: Numerically robust incremental algorithm for constructing three-dimensional Voronoi diagrams. *Proceedings of the 4th Canadian Conference on Computational Geometry*, St. John's, August 1992, pp. 334–339.

[21] Karasick, M., Lieber, D., and Nackman, L. R., 1989: Efficient Delaunay triangulation using rational arithmetic. IBM Report RC14455, IBM Thomas J. Watson Research Center, Yorktown Heights.

[22] Knuth, D. E., 1992: *Axioms and Hulls*. Lecture Notes in Computer Science, no. 606. Springer-Verlag, Berlin.

[23] Liotta, G., Preparata, F. P., and Tamassia, R., 1997: Robust proximity queries—an illustration of degree-driven algorithm design. *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pp. 156–165.

[24] Mehlhorn, K., and Näher, S., 1995: A platform for combinatorial and geometric computing. *Communications of the ACM*, January 1995, pp. 96–102.

[25] Milenkovic, V., 1988: Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, vol. 37, pp. 377–401.

[26] Minakawa, T., and Sugihara, K., 1997: Topology-oriented vs. exact arithmetic—experience in implementing three-dimensional convex hull algorithm. In H. W. Leong, H. Imai and S. Jain (eds.): *Algorithms and Computation, 8th International Symposium, ISAAC* '97, pp. 273–282. Lecture Notes in Computer Science, no. 1350, Springer-Verlag, Berlin.

[27] Oishi, Y., and Sugihara, K., 1995: Topology-oriented divide-and-conquer algorithm for Voronoi diagrams. *Computer Vision, Graphics, and Image Processing*: *Graphical Models and Image Processing*, vol. 57, pp. 303–314.

[28] Ottmann, T., Thiemt, G., and Ullrich, C., 1987: Numerical stability of geometric algorithms. *Proceedings of the 3rd Annual ACM Conference on Computational Geometry*, Waterloo, June 1987, pp. 119–125.

[29] Schorn, P., 1991: Robust algorithms in a program library for geometric computation. Dissertation submitted to the Swiss Federal Institute of Technology, Zürich, for the degree of Doctor of Technical Sciences, Informatik-Dissertationen ETH Zürich, Nr. 32.

[30] Segal, M., and Sequin, C. H., 1985: Consistent calculations for solid modeling. *Proceedings of the ACM Symposium on Computational Geometry*, Baltimore, June 1985, pp. 29–38.

[31] Shewchuk, J. R., 1996: Robust adaptive floating-point geometric predicates. *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, Philadelphia, May 1996, pp. 141–150.

[32] Steinitz, E., 1916: *Polyheder und Raumeinteilungen*. Encyklopädie der mathematischen Wissenschaften, Band III, Teil 1, 2. Hälfte, IIIAB12, pp. 1–139.

[33] Steward, A. J., 1994: Local robustness and its application to polyhedral intersection. *International Journal of Computational Geometry and Applications*, vol. 4, pp. 87–118.

[34] Sugihara, K., 1992: An intersection algorithm based on Delaunay triangulation. *IEEE Computer Graphics and Applications*, vol. 12, no. 2 (March 1992), pp. 59–67.

[35] Sugihara, K., 1993: Approximation of generalized Voronoi diagrams by ordinary Voronoi diagrams. *Computer Vision, Graphics, and Image Processing*: *Graphical Models and Image Processing*, vol. 55, pp. 522–531.

[36] Sugihara, K., 1994: Robust gift wrapping for the three-dimensional convex hull. *Journal of Computer and System Sciences*, vol. 49, pp. 391–407.

[37] Sugihara, K., 1994: A robust and consistent algorithm for intersecting convex polyhedra. *Proceedings of EUROGRAPHICS* '94, Oslo, September 1994, pp. c.45–c.54.

[38] Sugihara, K., 1997: Experimental study on acceleration of an exact-arithmetic geometric algorithm. *International Conference on Shape Modeling and Applications*, Aizu-Wakamatsu, March 1997, pp. 160–168.

[39] Sugihara, K., and Iri, M., 1988: Geometric algorithms in finite-precision arithmetic. *Abstracts of the* 13*th International Symposium on Mathematical Programming*, Tokyo, August–September 1988, WE/3K2, p. 196.

[40] Sugihara, K., and Iri, M., 1989: A solid modelling system free from topological inconsistency. *Journal of Information Processing*, vol. 12, pp. 380–393.

[41] Sugihara, K., and Iri, M., 1992: Construction of the Voronoi diagram for "one million" generators in single-precision arithmetic. *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1471–1484.

[42] Sugihara, K., and Iri, M., 1994: A robust topology-oriented incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry and Applications*, vol. 4, pp. 179–228.

[43] Yap, C., 1988: A geometric consistency theorem for a symbolic perturbation scheme. *Proceedings of the* 4*th Annual ACM Symposium on Computational Geometry*, Urbana-Champaign, June 1988, pp. 134–142.

[44] Yap, C., 1994: Exact computational geometry and tolerancing metrology. In D. Avis and P. Bose (eds.): *Snapshots of Computational Geometry*, vol. 3, pp. 34–48. Technical Report SOCS-94.50, McGill University.

[45] Zhu, X., Fang, S., Brüderlin, B. D., 1993: Obtaining robust Boolean set operations for manifold solids by avoiding and eliminating redundancy. *Proceedings of the* 2*nd Symposium on Solid Modeling and Applications*, Montreal, May 1993, pp. 147–154.