International Journal of Computational Geometry & Applications
Vol. 4, No. 1 (1994) pp 87–118

LOCAL ROBUSTNESS AND ITS APPLICATION TO POLYHEDRAL INTERSECTION

A. JAMES STEWART
Department of Computer Science,
University of Toronto, 10 Kings College Road,
Toronto, Ontario, Canada, M5S 1A4

Received 23 January 1992
Revised 19 October 1992
Communicated by C. M. Hoffmann

ABSTRACT

The field of solid modeling makes extensive use of a variety of geometric algorithms. When implemented on a computer, these algorithms often fail because the computer only provides floating point arithmetic, while the algorithms are expecting infinite precision arithmetic on real numbers. These algorithms are not *robust.*

This paper presents a formal theory of robustness. It is then argued that the elegant theoretical approach to robustness is not viable in practice: algorithms like those used in solid modeling are generally too complex for this approach. This paper presents a practical alternative to the formal theory of robustness; this alternative is called *local robustness*.

Local robustness is applied to the design of a polyhedral intersection algorithm, which is an important component in many solid modelers. The intersection algorithm has been implemented, and, in extensive tests, has never failed to produce a valid polyhedron of intersection. A concise characterization of the locally robust intersection algorithm is presented; this characterization can be used to develop variants of the intersection algorithm, and to develop robust versions of other solid modeling algorithms.

Keywords: numerical robustness, geometric algorithms

1. Introduction

Geometric algorithms are used in many important applications, such as computer aided design and finite element analysis. These algorithms, though provably correct in a formal sense, often fail when implemented on a computer.

This failure is not simply the fault of "bad programming." Failure occurs because the computer only provides floating point arithmetic where the program is expecting perfect arithmetic on real numbers. The absence of perfect arithmetic forces the program to make guesses where it should make accurate decisions, and often leads to failure.

Informally, an algorithm whose computer implementation doesn't fail is said to be *robust.* This paper presents both theoretical and practical approaches to the building of robust algorithms. A formal theory of robustness is presented, but it is argued that this theory is too rigorous to be used in practice. On the practical side of things, an alternative theory of “local robustness” is presented.

Local robustness is applied to the design of a polyhedral intersection algorithm. The intersection algorithm has been implemented, and, in over two hundred rigorous tests which would crash most intersection algorithms, has never failed to produce a valid polyhedron of intersection.

1.1. Practical Geometric Algorithms

There are many geometric algorithms with significant practical applications. For example, triangulation algorithms are used to generate finite element meshes,[1] which can be used in evaluating partial differential equations to compute such things as heat flow in solid objects and the performance of materials under physical stress. Other practical applications include graphics, which uses hidden line removal algorithms, computer vision, which uses convex hull and Voronoi diagram algorithms, and robot path planning, which uses polyhedral intersection algorithms to determine potential collisions.

Solid modelers — also known as CAD systems, for Computer Aided Design — are very useful in designing manufactured goods such as tools and automobiles. A solid modeler enables the designer to construct computer representations of objects, much like an electronic drafting board. The computer representation of an object can then be manipulated by other programs. For example, the computer representation of a shock absorber might be used by another program to simulate the action of the shock absorber on a bumpy road.

Many solid modelers use a “boundary representation” for their objects: each object is represented as a collection of faces, where each face is some mathematical surface. The simplest such surface is a plane, and the objects representable with planar faces are called *polyhedra*.

With a solid modeler, the designer can build an object by combining simpler objects. For example, the threadless machine nut of Figure 1 is created by intersecting a hexagonal slab with the complement of a solid, faceted cylinder.

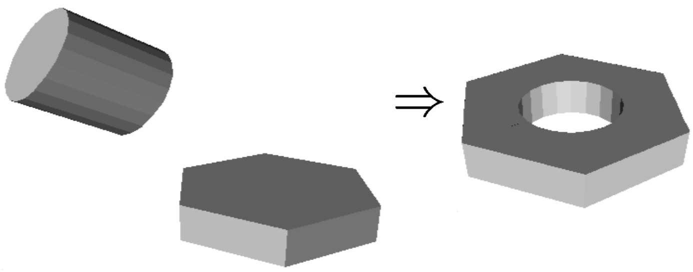

Fig. 1. Constructing a Machine Nut

A solid modeler usually has at least three basic operations with which to combine objects: intersection, union, and complementation. The complement of an object is trivial to compute, whereas the union of two objects can be computed using the complementation and intersection operations. Thus, intersection is a necessary operation, and a *polyhedral intersection algorithm* is of fundamental importance to solid modeling.

1.2. Why Do Geometric Algorithms Fail?

The input to an algorithm can be thought of as a sequence of bits $\overline{x}$, and the algorithm itself can be thought of as computing functions $f(\overline{x})$ of this input, then branching based upon the sign of $f(\overline{x})$ (see Ref. 2). For example, an algorithm can determine whether a point v is "below" an oriented plane P which has the equation $P(x) = 0$ by evaluating $P(v)$: if the sign of $P(v)$ is negative then the point is said to lie below the plane; if it is positive then the point is said to lie above the plane.

If an algorithm could compute with *real* arithemetic (on some *real* input $\overline{x}$), then it could always compute $f(\overline{x})$ exactly and would always make the correct branch. However, an algorithm that operates with floating point arithmetic cannot be certain of the sign of $f(\overline{x})$ when

$$|f(\overline{x})| \; < \; \varepsilon,$$

for some small constant ε. This situation is called an *ambiguous branch.*

The first cause of an ambiguous branch is *roundoff error*. A floating point number is represented with two components: a mantissa and an exponent. If two numbers, a and b, are represented with P-bit mantissas, then the product, ab, must in general be represented with a $2P$-bit mantissa. However, to keep the size of these numbers from growing, floating point arithmetic rounds the mantissa of ab to P bits. Thus, if a third number c is subtracted from ab the sign of $ab - c$ is not known when $|ab - c| \; < \; 2^{-P}\,|ab|$.

The second cause of an ambiguous branch is *inaccurate input*. For example, suppose a polyhedron is represented with a set of plane equations (one for each face) and some topological information. It is possible that the topological information specifies that four faces are to meet at a vertex, but that the actual faces (lying on planes having floating point coefficients) meet instead in a structure consisting of two vertices and a short edge,[3] as shown in Figure 2.

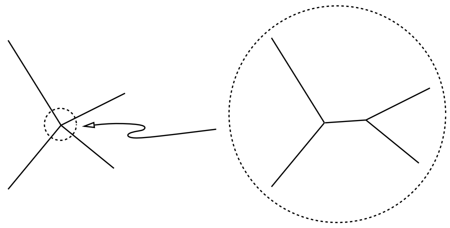

Fig. 2. Four Planes Meeting

If the four faces do indeed meet at two vertices, then we must assume that the coefficients of the plane equations are inaccurate by some small amount δ. Thus, when a point v is sufficiently close to a plane P, the algorithm cannot determine by any numeric test whether v lies on, above, or below P.

When presented with an ambiguous branch the algorithm will make an arbitrary decision. Most algorithms use the rule that

$$|\tilde{f}(\overline{x})| < \varepsilon \Rightarrow f(\overline{x}) = 0,$$

where $\tilde{f}$ is the computed value of f and ε is some small constant. This is the principle of "if it's close to zero, then it *is* zero" and is a common source of failure in many geometric algorithms.

For example, consider Figure 3. On line ℓ_1, the points a and b are "close" to the line ℓ_2 (indicated by their gray regions of uncertainty), and by the aforementioned principle are considered to be ON ℓ_2. Since ℓ_1 and ℓ_2 share the distinct points a and b, these two lines must be coincident. But this is contradicted by the point c, which is ON ℓ_1 and OFF ℓ_2. Thus an algorithm which uses the naive principle would make contradictory decisions in this situation. Such decisions often lead to failure.

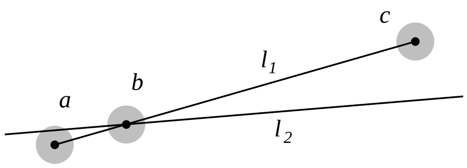

Fig. 3. Misclassifying Points

In summary, geometric algorithms that use floating point arithmetic are often faced with ambiguous branches. In such cases, the algorithm will usually make an arbitrary decision. If the algorithm makes a series of arbitrary decisions that are mutually contradictory, then the algorithm will probably fail. Robust algorithms do not make contradictory decisions.

We need a theory with which to build robust algorithms. Furthermore, this theory should be practical enough to be usable in the construction of real programs. Section 2 reviews the approaches taken by other researchers in building robust algorithms.

Section 3 presents a formal theory of robustness. It is argued that the formal notion of robustness is too rigorous for practical applications, and that an alternative approach — called *local robustness* — must be taken.

Section 4 contains the local robustness theorems concerning the polyhedral intersection algorithm. Section 5 discusses how these theorems are incorporated in a polyhedral intersection algorithm, and presents empirical tests of the algorithm.

2. Building Robust Geometric Algorithms

In this section we discuss the approaches taken by other researchers in building robust geometric algorithms. These approaches will be considered in three general categories: exact arithmetic methods, representation and model methods, and topological consistency methods. For other surveys of robustness research, the reader is directed to an article by Hoffmann[4] or to Chapter 4 of a text by Hoffmann.[5]

2.1. Exact Arithmetic Methods

Algorithms in this category operate on exact computer representations of objects. For example, a polygon might be represented by an ordered list of vertices, the coordinates of which are given as exact points on an integer grid.

Given some exact input $\overline{x}$, the algorithm can use high precision intermediate computations to compute the exact value of functions $f(\overline{x})$ of the input. Thus, an algorithm that uses exact input will never face an ambiguous branch, and will always succeed.

2.1.1. Milenkovic

Milenkovic[6] builds a polygonal modeling system that operates on "normalized polygons." These polygons obey a set of rules that allow the use of limited precision arithmetic in performing geometric tests. The process of converting an approximate polygon into a normalized polygon is called "accommodation" and involves shifting vertices and breaking edges to ensure a minimum feature separation, which in turn allow the use of limited precision arithmetic.

This approach is similar to that of Greene and Yao,[7] except that they shift the vertices onto grid points and do not explicitly use limited precision arithmetic.

2.1.2. Sugihara and Iri

Sugihara and Iri[8] propose a polyhedral modeling system that uses exact input and high precision intermediate computations. A two dimensional version of the modeling system has been implemented; the three dimensional polyhedral modeling system would allow an object to be defined by a set-theoretic combination of a restricted class of trihedral polyhedra whose faces lie on a restricted set of exactly represented planes.

The modeling system would allow rotations and translations to be applied to primitive objects. However, when such objects are transformed, their plane equations must generally be represented with more precision than before. To prevent an explosive growth in the required precision, the primitive objects must be represented approximately (that is, rounded). However, a later result of Milenkovic and Nackman[9] shows that, in general, trihedral polyhedra cannot be rounded efficiently without altering the topological structure. Thus, if we use exactly represented polyhedra, then we must find a restricted set of "roundable" rotations for these polyhedra.

Other work in the exact arithmetic category includes that of Ottmann, Thiemt, and Ullrich,[10] who build a robust algorithm to report line segment intersections.

2.2. Representation and Model Methods

Algorithms in this category operate on approximate computer representations of objects. For example, a line can be represented by a line equation $ax+by+c=0$, where the coefficients a, b, and c are allowed to vary within a small range. Thus an approximate representation of a line could denote a whole family of real lines.

At an abstract level, these algorithms appear to operate as follows: given an approximate representation of the input object (or objects), the algorithm chooses one *real* object from the family of real objects denoted by the approximate representation, and computes on that real object. The real object is called the *model* and is never explicitly represented by the algorithm.

In reality, these algorithms make a series of decisions — and corresponding branches — that are consistent with one particular model M of the input. As long as ambiguous branches are decided in this consistent manner, the algorithm will not fail; it will make the same decisions as would an exact algorithm that operates on the exactly represented model M.

The difficulty in the implict model approach lies in proving that a model always exists which corresponds to the decisions of the algorithm. This is often a nontrivial task and is perhaps why the exact arithmetic approach is more popular with the implementors of geometric algorithms.

2.2.1. Hoffmann, Hopcroft, and Karasick

Hoffmann, Hopcroft, and Karasick[3] introduce the representation and model idea. They define an algorithm to be *correct* (or robust, in the terminology of this article) if it computes a mapping from input representations R_i to output representations R_o such that there exist corresponding models M_i and M_o for which this is the correct mapping.

The problem of intersecting two polygons is considered, and it is shown that any vertex and edge that are "close" in the polygon representations can be considered coincident in the model polygons, for there will always exist some real model polygons in which all such incidences are realized. The model polygons are not necessarily simple, nor are their edges necessarily close to the edges of the polygon representation. Thus, the intersection algorithm does not guarantee a bound on the perturbation of the model.

The problem of simultaneously intersecting three polygons is considered, and it is shown that this is sufficiently hard that it requires a thoerem prover for projective geometry. It is also shown that the problem of arranging the vertices of two polyhedra is at least as difficult as that of simultaneously intersecting three polygons.

2.2.2. Hopcroft and Kahn

Hopcroft and Kahn[11] present the first provably robust algorithm for intersecting an approximately represented convex polyhedron with a halfspace. Any vertices of the polyhedron that are close to the plane defining the halfspace are placed ON the plane. It is proven that there always exists a real polyhedron with the same incidences; however, this real polyhedron could be arbitrarily distant from the polyhedron representation.

This result could be used to build an algorithm to intersect two approximately represented convex polyhedra. However, the algorithm would not necessarily be robust in the representation and model sense (see the discussion in their article).

2.2.3. Milenkovic

Milenkovic[6,12] presents a provably robust algorithm that computes the arrangement of a set of pseudolines. The input consists of a set of lines with P-bit coefficients. The algorithm computes the arrangement of a set of implicit pseudolines that lie within a distance 2^{-P} of the lines. An important feature of the algorithm is that only $P+5$ bits are required for intermediate computations.

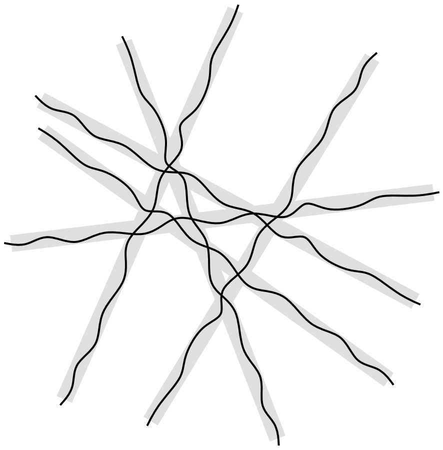

Fig. 4. An Arrangement of Pseudolines Within Fixed Bounds

In later work, Milenkovic extends this to a more general approach called *double precision geometry*.[13] This approach can be used to make robust any algorithm that calculates line and segment arrangements, as long as the algorithm can handle an arrangement of monotonic pseudolines instead of an arrangement of lines. The computed arrangement differs from the real line arrangement only in areas where vertices are close. For input lines that are represented with P-bit coefficients, the algorithm uses $2P+5$ bits for intermediate computations. In other work, this is extended to computing arrangements of planar curves, instead of pseudolines.[14]

2.2.4. Fortune

Fortune[15] considers robustness in a manner similar to Hoffman, Hopcroft, and Karasick: an algorithm is *robust* if it always computes the solution to some real problem instance, and an algorithm is *stable* if the real problem instance is close to the given problem instance. This definition assumes exact input and inexact intermediate computations, although it could be easily modified for approximate input.

Fortunes presents two robust algorithms. The first computes the convex hull of an implicit set of n points perturbed by at most $\mathcal{O}(n\varepsilon)$ from a set of input points, where ε is approximately the rounding error. The second algorithm maintains a triangulation of an implicit set of n points which are perturbed at most $\mathcal{O}(n^2\varepsilon)$ from a set of input points.

2.2.5. Salesin, Stolfi, and Guibas

Salesin, Stolfi, and Guibas[16] and Salesin[17] present an approach to robustness called *epsilon geometry*, in which an algorithm computes on some perturbed version of the input. An interesting aspect of this work is that algorithms based upon epsilon geometry compute, dynamically, a bound on the perturbation.

Under this approach, an algorithm evaluates predicates $P(x)$ on the input x that compute, instead of TRUE or FALSE, an uncertainty interval $[l, h]$: no perturbation of x by less than distance l can make $P(x)$ true, and some perturbation of x by more than distance h can make $P(x)$ true (for some appropriate definition of "distance"). Predicates can be combined with the standard logical operations, and rules are given for combining their associated uncertainty intervals. Designing algorithms under this approach involves extensive manipulation of these uncertainty intervals.

Salesin[17] demonstrates this approach in implementing robust algorithms for point location in an approximately convex polygon, for testing approximate and strong convexity of polygons, and for constructing ε-convex δ-hulls of sets of points (similar to the convex hulls of Li and Milenkovic[18]).

2.3. Topological Consistency Methods

Algorithms in this category maintain certain topological properties of the objects that they manipulate. For example, a topologically consistent line arrangement algorithm would never decide that two lines intersect more than once, and a topologically consistent Voronoi diagram algorithm would ensure that the resulting Voronoi diagram has a planar embedding.

In general, algorithms in this category deal with much harder problems than those in the other two categories. Algorithms in this category are usually *not* proved robust in the representation and model sense.

2.3.1. Sugihara

Sugihara eliminates redundant numeric tests from his algorithms. The algorithms are not proved robust in the representation and model sense, but empirical tests show them to be immune from numerical error.

Sugihara and Iri[19,20] present a Voronoi diagram algorithm that is guaranteed to run to completion and to output a planar graph even if every numeric test is decided randomly (simulating 0-bit arithmetic). Obviously, the output in such a case is not too meaningful, but as the precision of the intermediate computations increases, the output of the algorithm converges to the true Voronoi diagram. Other work includes that of Sugihara[21] and Sugihara, Ooishi, and Imai.[22]

2.3.2. Brüderlin

Brüderlin[23,24] builds a polyhedral intersection algorithm that maintains a region of uncertainty around each feature. In a similar manner to Segal and Séquin[27] (see below) his algorithm decides that close features are coincident, and merges their corresponding regions of uncertainty.

However, Brüderlin's algorithm keeps sufficient information about the regions of uncertainty to determine when its decisions to merge close features are potentially inconsistent. It is proven that if a potentially inconsistency is *not* detected, then there exists a model (in the "representation and model" sense) which corresponds to the decisions of the algorithm. However, if a potential inconsistency *is* detected, then the algorithm must be re-executed with smaller regions of uncertainty (and possibly more precision). Brüderlin states that, in practice, the algorithm rarely has to be run more than twice.

2.3.3. Karasick

Karasick[28] presents a polyhedral intersection algorithm that incorporates some *ad hoc* robustness rules (a concise description of this work is given by Hoffman, Hopcroft, and Karasick[29]). For example, one such rule states that a vertex v of one polyhedron cannot lie ON a face f of another polyhedron if the vertex is on an edge e that intersects that face at a distinct point $e \cap f \neq v$. This type of rule could prevent the inconsistent decisions that might be made in Figure 3, as described in Section 1.2.

Two empirical tests were introduced by Karasick. The first test involved repetitively intersecting an object with a rotated version of itself: each intersection approximately doubled the number of faces in the object. The second test involved intersecting two superimposed cubes, one slightly rotated from the other, and finding the range of angles in which the algorithm failed. These tests are discussed in detail in Section 5.3 and Section 5.4.

No proof of robustness is offered for Karasick's intersection algorithm. However, for cube/cube test the algorithm failed in a very small range of angles, and for the repetitive intersection test the program ran out of memory before it could fail.

2.3.4. Segal and Séquin

Segal and Séquin[27] develop a polyhedral intersection algorithm that maintains a region of uncertainty around each feature of the polyhedra. When features are close, they are merged and the regions of uncertainty are grown. If these regions grow too large a "tolerance error" is signalled to the user. Also, if a test about a vertex/plane relationship is required and tolerance regions of the vertex and the plane overlap, then a "tolerance error" is signalled and an arbitrary decision is made.

In another approach, Segal[30] incorporates four robustness criteria into his polyhedral intersection algorithm. When the algorithm discovers a configuration of features that contradicts the criteria, it backtracks, undoing decisions based on numeric data, until the contradiction is eliminated. While no proof of robustness is given, a cube/cube test (similar to that of Karasick) reveals *no* region of error in Segal's algorithm.

2.3.5. Stewart

Like the work of Karasick and of Segal and Séquin, the author's polyhedral intersection algorithm incorporates certain rules to promote robustness. The author's work differs in that the rules are developed with a particular methodology (see Section 3.2) and proofs are presented which show that an algorithm incorporating these rules will not fail in particular modes (see Section 4.3). The methodology, called "local robustness," is applicable to other geometric problems.

2.4. Other Related Work

Dobkin and Silver [32] demonstrate how numeric error can accumulate in cascaded computations, where the output of one computation is used as input by the next.

A "permutation/perturbation method" of avoiding numeric failure is suggested by Dobkin and Silver. Using this approach, the algorithm is run with various permutations of its numeric code and the results of the permutations are compared to find an average, empirical error in the computations. When the error becomes too large, the algorithm increases arithmetic precision and backtracks to some earlier point.

Edelsbrunner and Mücke [34] present a technique — called "simulation of simplicity" — by which an algorithm can avoid considering degenerate cases of the input. A similar technique was simultaneously presented by Yap [35].

Using simulation of simplicity, the input, which is assumed to be exactly represented, is perturbed infinitesimally so that *every* numeric sign test which might return $+1$, -1, or 0 instead only returns $+1$ or -1. This perturbation is accomplished by a "black box" and is transparant to the algorithm.

This technique is particularly useful, since a large amount of the time spent on algorithm design is usually spent considering the degenerate cases of the input. For example, a Voronoi diagram algorithm that uses this approach could ignore the special case in which four points lie on the same circle.

However, this approach is not immediately applicable to such problems as solid modeling, in which degenerate cases are often necessary: consider that four vertices on the same face of a polyhedron are in a degenerate configuration.

3. Correctness, Robustness, and Local Robustness

Consider the problem of multiplying two real numbers. There are three components involved: the pair of real numbers to be multiplied (called the *problem instance*), the real number that is the product (called the *problem solution*), and the mapping from instances to solutions (simply called the *problem*).

A problem maps instances onto solutions,

$$\mathcal{P} : \mathcal{I} \rightarrow \mathcal{S}.$$

An *algorithm* is a method by which to compute the problem mapping. When an algorithm is executed on a computer it must use *representations* for both the problem instance and the problem solution. An algorithm maps representations of problem instances onto representations of problem solutions,

$$A : \mathcal{I}_{\text{rep}} \rightarrow \mathcal{S}_{\text{rep}}.$$

For a given problem and algorithm, there are natural one-to-one mappings between the abstract entities and their corresponding representations (one-to-many mappings are discussed shortly). For example, the computer representation of an integer maps onto a real, mathematical integer. The mappings that concern us are

$$\begin{aligned} \text{MODEL} : \quad & \mathcal{I}_{\text{rep}} \rightarrow \mathcal{I}, \quad \text{and} \\ \text{REP} : \quad & \mathcal{S} \rightarrow \mathcal{S}_{\text{rep}}. \end{aligned}$$

This is summarized in the following diagram:

$$\begin{array}{ccccc} & \mathcal{I} & \xrightarrow{\mathcal{P}} & \mathcal{S} & \\ \text{MODEL} & \uparrow & & \downarrow & \text{REP} \\ & \mathcal{I}_{\text{rep}} & \xrightarrow{A} & \mathcal{S}_{\text{rep}} & \end{array}$$

An algorithm A for a problem $\mathcal{P}$ is *correct* if the above diagram commutes; that is, if for all $x_{rep} \in \mathcal{I}_{\text{rep}}$,

$$A(x_{rep}) = \text{REP}(\ \mathcal{P}(\ \text{MODEL}(x_{rep})\ )\ ).$$

An *approximate* representation (as opposed to an *exact* representation) is one that denotes a set of abstract entities. For example, the floating point encoding of a real number is an approximate representation, since each floating point number denotes a whole family of real numbers. For a given approximate representation, there is a natural mapping from approximate problem instances onto sets of real problem instances,

$$\text{MODELS} : \mathcal{I}_{\text{rep}} \Rightarrow \mathcal{I},$$

where the large arrow indicates a one-to-many mapping. This is summarized in the following diagram:

$$\begin{array}{ccccc} & \mathcal{I} & \xrightarrow{\mathcal{P}} & \mathcal{S} & \\ \text{MODELS} & \Uparrow & & \downarrow & \text{REP} \\ & \mathcal{I}_{\text{rep}} & \xrightarrow{A} & \mathcal{S}_{\text{rep}} & \end{array}$$

An algorithm A for a problem $\mathcal{P}$ is *robust* if, for every approximate problem instance $x_{rep} \in \mathcal{I}_{\text{rep}}$, there is a real problem instance $x \in \text{MODELS}(x_{rep})$ such that

$$A(x_{rep}) = \text{REP}(\ \mathcal{P}(x)\ ).$$

This definition of robustness is based upon the representation and model approach of Hoffmann, Hopcroft, and Karasick.[3] It is similar to the definition of Fortune.[15] The representation and model approach is similar to Milenkovic's hidden variable approach.[6]

3.1. Local Robustness

That some geometric problems are inherently difficult to solve robustly was concluded by Milenkovic.[6] He states that "all true line arrangements must satisfy a rich set of theorems of projective geometry" and illustrates this with the example of Pappus's Theorem from projective geometry. The conclusion of Milenkovic is that a robust line arrangement algorithm must embody a large set of theorems of geometry, and that it is probably unreasonable to expect to build such an algorithm having less than exponential time complexity.

Furthermore, Hoffmann, Hopcroft, and Karasick[3] show that the problem of simultaneously intersecting three polygons and the problem of arranging the vertices of two polyhedra are sufficiently hard that they requires a thoerem prover for projective geometry. If polyhedral intersection is as hard as polyhedral arrangement (as would appear to be the case), then it, too, would require a theorem prover for projective geometry.

Thus, there are some practical problems for which it is unlikely that an efficient, robust algorithm exists.

Define a *feature* to be a point, a line, or a plane. Two approximately represented features are *adjacent* if they have a model in which one is embedded in the other. A set of features is *adjacent* if the corresponding undirected graph (under the "adjacent" relation) is connected. A set of adjacent features is *small* if the diameter of the corresponding graph is small.

The author has observed that nonrobust geometric algorithms almost always fail due to an inconsistent set of decisions about a small set of adjacent features. Such "local failures" can be avoided if the algorithm ensures that ambiguous relations in small sets of adjacent features are interpreted consistently with some model of those features.

Local Robustness
An algorithm is *locally robust* for a set of features if it is robust for all inputs consisting of exactly those features.

Since this is usually the *only* kind of failure in nonrobust geometric algorithms, we should be willing to believe that a locally robust algorithm will probably never fail.

Principle of Local Robustness
If an algorithm is locally robust for many small sets of adjacent features, then it is unlikely that the algorithm will fail in practice.

Local robustness is a weaker criterion than robustness, and should be used in its place whenever it is not possible to build an efficient, robust algorithm. Unlike robustness, which guarantees that a global model exists which corresponds to the decisions of the algorithm, local robustness only guarantees that local models exist for various small sets of adjacent features which are chosen by algorithm designer.

3.2. Implementing Local Robustness

In developing locally robust algorithms, the reasoning process is specific to each problem (just as it is with robust algorithms). For any given problem, a simple methodology helps in developing a locally robust algorithm:

1. Consider small sets of adjacent features in various problem instances.
2. For each such set, determine "critical properties" that characterize it.
3. Build an algorithm whose decisions are consistent with these properties.

For example, consider a line arrangement algorithm, for which the input consists of the approximate locations of a set of lines and the output consists of a symbolic description of the arrangement. In a line arrangement, one small set of adjacent features is a pair of lines and their point of intersection. A critical property of a pair of lines is that they intersect at most once. A locally robust arrangement algorithm which incorporates this critical property will output an arrangement of pseudo-lines. Such an arrangement of pseudo-lines is just as useful to the majority of practical applications as an arrangement of straight lines, and, as Milenkovic demonstrated,[6] can be computed robustly and efficiently.[a]

Consider a second example: Sugihara and Iri[19,20] propose two important properties that a Voronoi diagram algorithm must preserve:

(P1) the Voronoi diagram is a planar graph that partitions the plane into as many cells as the generators, and

(P2) two cells share at most one common edge.

Their algorithm incrementally add points to a Voronoi diagram, each time modifying the Voronoi diagram in a local area and preserving these properties within that area. The algorithms of Sugihara and Iri[19,20] and of Sugihara, Ooishi, and Imai[22] are remarkable in their ability to withstand numerical error.

In summary, the following three concepts have been defined:

- An algorithm operating on an exact representation is *correct* if it computes the correct mapping.
- An algorithm operating on an approximate representation is *robust* if, given an approximate problem instance, the algorithm computes the correct mapping for any one of the corresponding real problem instances.

[a]Milenkovic's algorithm is robust in that it always computes the arrangement of some set of pseudo-lines. However, if the output is considered to be an arrangement of lines (which was *not* Milenkovic's intention), then the algorithm is only locally robust.

- An algorithm operating on an approximate representation is *locally robust* for some set of features if it is robust for all inputs consisting of exactly those features.

4. Locally Robust Polyhedral Arrangement

It is an interesting open problem whether a polyhedral intersection algorithm must, in general, compute the arrangement of the planes defined by the faces of the polyhedra being intersected.

If this is indeed the case, then it is is unlikely that a robust and efficient polyhedral intersection algorithm exists: plane arrangement is at least as hard as line arrangement and Milenkovic[6] and Hoffmann, Hopcroft, and Karasick[3] show that line arrangement requires a theorem prover for projective geometry.

In this section a variant of the plane arrangement problem will be considered and criteria will be developed for a locally robust algorithm which solves this problem. The methodology of Section 3.2 will be applied: critical properties will be determined for small sets of adjacent features in several different problem instances.

4.1. The Polyhedral Arrangement Problem

We consider a variant of the plane arrangement problem which is useful for polyhedral intersection. This variant is called the *polyhedral arrangement problem.*

Each face, edge, and vertex of a polyhedron corresponds to some plane, line, or point in $\mathbb{R}^3$. A plane p has an associated normal $\hat{p}$ which defines a halfspace h_p^+ "above" the plane (which contains the outward pointing normal) and a halfspace h_p^- "below" the plane. A line ℓ has an associated direction $\hat{\ell}$ which defines the ordering of points along the line.

Relations exist among these features; the relations that we will consider are summarized here:

- $\textsc{incidence}(a, b) \in \{\textsc{on}, \textsc{off}\}$. A point a is either ON or OFF a point b.
- $\textsc{location}(a, p) \in \{\textsc{below}, \textsc{on}, \textsc{above}\}$. A point a is either ON a plane p, or is BELOW it ($a \subset h_p^-$) or ABOVE it ($a \subset h_p^+$).
- $\textsc{order}(a, b, \ell) \in \{<, =, >\}$. A point a either precedes, is incident to, or follows a point b on a directed line ℓ. The relation is only defined when both a and b are ON ℓ.
- $\textsc{direction}(\ell, p) \in \{\textsc{negative}, \textsc{on}, \textsc{positive}\}$. A directed line ℓ is either ON a plane p, or pierces the plane such that the dot product $\hat{p} \cdot \hat{\ell}$ is NEGATIVE or POSITIVE.
- $\textsc{side}(a, \ell, p) \in \{\textsc{left}, \textsc{on}, \textsc{right}\}$. A point a is either ON the directed line ℓ, or is to its LEFT or RIGHT with respect to the plane p that is shared by a and ℓ. For example, consider Figure 5: the cross product $\hat{p} \times \hat{\ell}$ points to the LEFT of ℓ with respect to p (assuming a right-handed coordinate system).

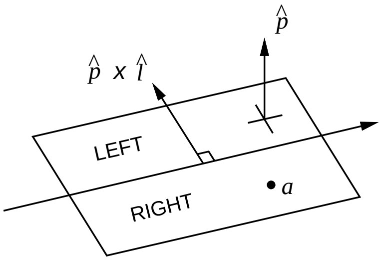

Fig. 5. SIDE(a, ℓ, p) = RIGHT

In the following discussion, the notation "a ON b" will mean INCIDENCE(a, b) = ON, the notation "$a <_\ell b$" will mean ORDER$(a, b, \ell) = <$, and other relations will be written similarly.

Recall that a *feature* is a point, a line, or a plane. Then an *instance* of the polyhedral arrangement problem consists of a set F of features, a set R_{allow} of all "allowable" relations between the features, and possibly some topological constraints relating the features.

Define a *complete* set of relations among the features of F as a set that contains exactly one relation for every group of features in F that can be related. Then a *solution* of the polyhedral arrangement problem consists of any complete set of relations $R \subseteq R_{allow}$ for which there exists a set of real features, corresponding to those of F, that satisfy the relations of R and the topological constraints, if any. Call this set of real features the *model* and call R an *arrangement*. We shall assume that R_{allow} always admits a valid R.

4.2. An Example of Polyhedral Arrangement

Say the problem instance consisted of two planes, their line of intersection, and a point. Then

$$F = \{a, p, q, \ell\},$$

where a is the point, p and q are the planes, and $\ell = p \cap q$. The set of allowable relations, R_{allow}, can be some subset of

$$\begin{aligned} \{ \quad & a \text{ BELOW } p, \quad a \text{ ON } p, \quad a \text{ ABOVE } p, \\ & a \text{ BELOW } q, \quad a \text{ ON } q, \quad a \text{ ABOVE } q, \\ & \ell \text{ ON } p, \quad \ell \text{ ON } q, \\ & a \text{ LEFT}_p\ \ell, \quad a \text{ ON}_p\ \ell, \quad a \text{ RIGHT}_p\ \ell, \\ & a \text{ LEFT}_q\ \ell, \quad a \text{ ON}_q\ \ell, \quad a \text{ RIGHT}_q\ \ell \ \}, \end{aligned}$$

where, for example, "a LEFT$_p$ ℓ" means SIDE(a, ℓ, p) = LEFT.

The relation between two features is unambiguous if and only if the same relation is true in all models of the two features. If features are represented by "regions of uncertainty", as is often the case,[6,14,24,27] then the relation between two features is unambiguous exactly when their regions of uncertainty do not intersect. Thus the construction of R_{allow} depends upon the representation of the polyhedra.

Suppose that an unambiguous numeric test determines that a is BELOW p. Then the set R_{allow} would be defined as

$$
\begin{array}{rl}
R_{allow} = \{ & a \text{ BELOW } p, \\
& a \text{ BELOW } q, \quad a \text{ ON } q, \quad a \text{ ABOVE } q, \\
& \ell \text{ ON } p, \quad \ell \text{ ON } q, \\
& a \text{ LEFT}_p\ \ell, \quad a \text{ ON}_p\ \ell, \quad a \text{ RIGHT}_p\ \ell, \\
& a \text{ LEFT}_q\ \ell, \quad a \text{ ON}_q\ \ell, \quad a \text{ RIGHT}_q\ \ell \ \},
\end{array}
$$

and a solution to the polyhedral arrangement problem would be any complete set $R \subseteq R_{allow}$ that has a model. For example, the set

$$R_{bad} = \{ \ a \text{ BELOW } p, \ \ a \text{ BELOW } q, \ \ \ell \text{ ON } p, \ \ \ell \text{ ON } q, \ \ a \text{ ON}_p\ \ell, \ \ a \text{ ON}_q\ \ell \ \}$$

would not be a solution.

4.3. Critical Properties for Polyhedral Arrangement

Using the methodology of Section 3.2, critical properties will be developed for a locally robust polyhedral arrangement algorithm. Several situations arising from the intersection of two polyhedra will be considered.

4.3.1. Intersection of Two Edges

Consider the intersection of two noncollinear polyhedron edges. The features of interest are two lines corresponding to the two edges, the point of intersection of the two lines, any other points that lie on the lines, and the plane that embeds the two lines. Then

$$F = \{p, \ell_a, \ell_b, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m, c\}.$$

where p is the plane, ℓ_a and ℓ_b are the lines, a_i and b_i are points ON ℓ_a and ℓ_b, respectively, and c is the point of intersection. The relations of interest are ORDER and SIDE. Assuming that no two points are coincident,

$$
\begin{array}{rlll}
R_{allow} = \{ & \ell_a \text{ ON } p, & \ell_b \text{ ON } p, & \\
& a_i \text{ ON}_p\ \ell_a, & b_i \text{ ON}_p\ \ell_b, & \\
& c \text{ ON}_p\ \ell_a, & c \text{ ON}_p\ \ell_b, & \\
& a_i <_{\ell_a} a_j, & a_i >_{\ell_a} a_j, & i \neq j, \\
& a_i <_{\ell_a} c, & a_i >_{\ell_a} c, & \\
& b_i <_{\ell_b} b_j, & b_i >_{\ell_b} b_j, & i \neq j, \\
& b_i <_{\ell_b} c, & b_i >_{\ell_b} c, & \\
& c <_{\ell_a} a_i, & c >_{\ell_a} a_i, & \\
& c <_{\ell_b} b_i, & c >_{\ell_b} b_i, & \\
& a_i \text{ LEFT}_p\ \ell_b, & a_i \text{ RIGHT}_p\ \ell_b, & \\
& b_i \text{ LEFT}_p\ \ell_a, & b_i \text{ RIGHT}_p\ \ell_a & \}
\end{array}
$$

where the subscripts i and j take on all possible values (that is, $1 \ldots n$ for a_i and a_j, and $1 \ldots m$ for b_i and b_j).

As stated in Theorem 1 below, the critical properties of Table 1 characterize those complete sets $R \subseteq R_{allow}$ that have models. In this table, a, b, and c are points, ℓ, ℓ_a, and ℓ_b are lines, and p is a plane. CP 1 and CP 2 preserve antisymmetry and transitivity for points ordered along a line. CP 3 ensures consistent SIDE and ORDER relations (see Figure 6).

Table 1. Critical Properties For Intersecting Two Edges

CP 1 :	$a <_\ell b \;\Leftrightarrow\; b >_\ell a$
CP 2 :	$a <_\ell b \;\wedge\; b <_\ell c \;\Rightarrow\; a <_\ell c$
CP 3 :	Let a, c ON_p ℓ_a, b, c ON_p ℓ_b, $\neg(a$ ON_p $\ell_b)$, $\neg(b$ ON_p $l_a)$. Then $\text{ORDER}(c, a, \ell_a) = \text{ORDER}(c, b, \ell_b) \;\Leftrightarrow\; \text{SIDE}(a, \ell_b, p) \neq \text{SIDE}(b, \ell_a, p)$

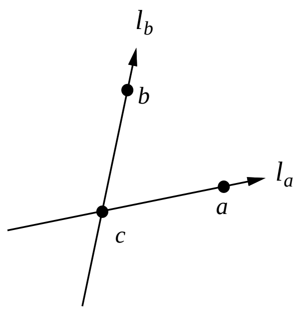

Fig. 6. Example of CP 3

Theorem 1 *Let F and R_{allow} be as defined above for the intersection of two edges. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a model if and only if R obeys CP 1 through CP 3.*

Proof.
(Only if) By inspection, the properties are true of every model.
(If) Construct a model for R as follows. Place ℓ_a and ℓ_b in general position on p and define $c = \ell_a \cap \ell_b$. Orient ℓ_a and ℓ_b arbitrarily. By CP 1 and CP 2 the points on ℓ_a are totally ordered, since every pair of points is related in R. Choose any set of points that satisfy this total order as a model for the points on ℓ_a. Do likewise for points on ℓ_b. Orient p to satisfy the relation $\text{SIDE}(a_1, \ell_b, p)$ in R. Then by CP 3 all $\text{SIDE}(b_i, \ell_a, p)$ relations in R are satisfied by this model (consider instantiating a to a_1 and b to b_i in CP 3). Similarly, all $\text{SIDE}(a_i, \ell_b, p)$ relations are satisfied by this model (consider instantiating a to a_i and b to b_1 in CP 3). $\square$

4.3.2. Intersection of Two Faces

Consider the intersection of two polyhedral faces in general position. The features of interest are two planes corresponding to the two faces, a line corresponding to their intersection, and many points corresponding to the vertices of the faces. Then

$$F = \{p_a, p_b, \ell, a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m\},$$

where p_a and p_b are two planes, ℓ is a line, a_i is a point ON p_a, and b_i is a point ON p_b. The relations of interest are LOCATION and SIDE. Assuming that no point lies on the intersection of the two planes,

$$\begin{array}{rll} R_{allow} = \{ & a_i \text{ ON } p_a, & b_i \text{ ON } p_b, \\ & \ell \text{ ON } p_a, & \ell \text{ ON } p_b, \\ & a_i \text{ BELOW } p_b, & a_i \text{ ABOVE } p_b, \\ & b_i \text{ BELOW } p_a, & b_i \text{ ABOVE } p_a, \\ & a_i \text{ LEFT}_{p_a}\ \ell, & a_i \text{ RIGHT}_{p_a}\ \ell, \\ & b_i \text{ LEFT}_{p_b}\ \ell, & b_i \text{ RIGHT}_{p_b}\ \ell\}. \end{array}$$

A new critical properties is introduced in Table 2, in which a and b are points, ℓ is a line, and p_a and p_b are planes. CP 4 ensures consistent LEFT and RIGHT labelling of points that lie on intersecting planes. See Figure 7, in which the outward pointing normals of the planes are shown as arrows.

Table 2. Critical Property for Intersecting Two Faces

CP 4 : Let a ON p_a, b ON p_b, $\neg(a$ ON $p_b)$, $\neg(b$ ON $p_a)$, $\ell = p_a \cap p_b$. Then $\text{LOCATION}(a, p_b) = \text{LOCATION}(b, p_a) \Leftrightarrow \text{SIDE}(a, \ell, p_a) \neq \text{SIDE}(b, \ell, p_b)$

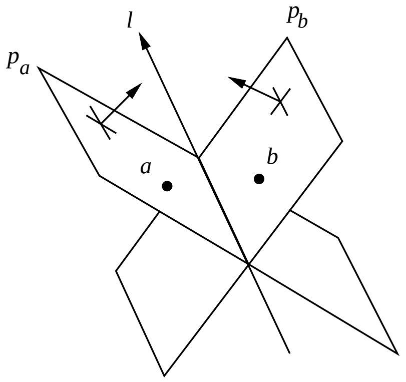

Fig. 7. Example of CP 4

Theorem 2 *Let F and R_{allow} be as defined above for the intersection of two faces. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a model if and only if R obeys CP 4.*

Proof.
(Only if) By inspection, the properties are true of every model.
(If) Construct a model for R as follows. Place two oriented planes, p_a and p_b, in general position and define $\ell = p_a \cap p_b$. Place each point x according to the $\text{LOCATION}(x, p_a)$ and $\text{LOCATION}(x, p_b)$ relations in R. Orient ℓ to satisfy the relation $\text{SIDE}(a_1, \ell, p_a)$ in R. Then by CP 4 all $\text{SIDE}(b_i, \ell, p_b)$ relations in R are satisfied by this model (consider instantiating a to a_1 and b to b_i in CP 4). Similarly, all $\text{SIDE}(a_i, \ell, p_a)$ relations are satisfied by this model (consider instantiating a to a_i and b to b_1 in CP 4). □

4.3.3. Intersection of an Edge and Many Faces

Consider the intersection of an edge of one polyhedron with many of the faces of another polyhedron. The features of interest are a line which corresponds to the edge, a plane for each face that intersects the edge, and a point at each intersection of the edge and a face. The endpoints of the edge will be ignored, as will the points corresponding to vertices of the faces. Then

$$F = \{\ell, p_1, p_2, \ldots, p_n, a_1, a_2, \ldots, a_n\},$$

where ℓ is the line, p_i are the planes, and $a_i = \ell \cap p_i$ are the points. The relations of interest are LOCATION, ORDER, and DIRECTION. Assuming that no points are coincident and that no plane embeds the line,

$$\begin{array}{lllll} R_{allow} = \{ & a_i \text{ ON } p_i, & & & \\ & a_i <_\ell a_j, & a_i >_\ell a_j, & i \neq j, & \\ & a_i \text{ BELOW } p_j, & a_i \text{ ABOVE } p_j, & i \neq j, & \\ & \ell \text{ POSITIVE } p_i, & \ell \text{ NEGATIVE } p_i & \}. & \end{array}$$

Four new critical properties are introduced in Table 3. Here a and b are points, ℓ is a line, and p is a plane. CP 5 through CP 8 are all similar: they ensure correct ABOVE and BELOW labeling of points with respect to planes (see Figure 8).

Table 3. Additional Critical Properties for Edge / Face Intersections

CP 5 :	$a \text{ ON } p \;\wedge\; a <_\ell b \;\wedge\; \ell \text{ POSITIVE } p \;\Rightarrow\; b \text{ ABOVE } p$
CP 6 :	$a \text{ ON } p \;\wedge\; a <_\ell b \;\wedge\; \ell \text{ NEGATIVE } p \;\Rightarrow\; b \text{ BELOW } p$
CP 7 :	$a \text{ ON } p \;\wedge\; a >_\ell b \;\wedge\; \ell \text{ POSITIVE } p \;\Rightarrow\; b \text{ BELOW } p$
CP 8 :	$a \text{ ON } p \;\wedge\; a >_\ell b \;\wedge\; \ell \text{ NEGATIVE } p \;\Rightarrow\; b \text{ ABOVE } p$

Theorem 3 *Let F and R_{allow} be as defined above for the intersection of an edge and many faces. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a model if and only if R obeys CP 1, CP 2, and CP 5 through CP 8.*

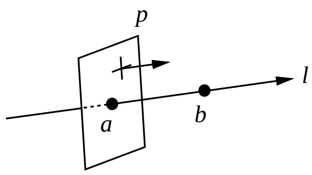

Fig. 8. Example of CP 5

Proof.
(Only if) By inspection, the properties are true of every model.
(If) Construct a model for R as follows. Choose an arbitrary line as a model for ℓ. Choose as models for the points a_i any set of points on that line that satisfy the ORDER relations in R; by the argument of Theorem 1 (using CP 1 and CP 2) there exist such points. Choose as a model for each plane p_i any oriented plane passing through a_i which satisfies the relation DIRECTION(ℓ, p_i) in R. Then by CP 5 through CP 8 the relation LOCATION(a_j, p_i) in R is satisfied for all other points a_j. □

4.3.4. Arrangements of a Corner

Consider arranging a vertex and a polyedron corner. The vertex lies BELOW or ABOVE each of the n faces of the corner. Does a model exist for each of the 2^n possible arrangements? This is is an interesting problem because an arbitrary set of planes cannot be chosen as a model, since the faces which lie on these planes are constrained to form a cone. A *cone* consists of a set of faces that meet at a common vertex, and is defined by a vertex and a sequence of at least three faces,

$$\mathcal{C} = \{v, f_1, f_2, \ldots, f_n\}.$$

Each semi-infinite face of a cone is bounded by two edges. A pair of faces share an edge exactly when they are adjacent in the sequence. All other pairs of faces share exactly the vertex v. The face normals are oriented consistently; that is, the ABOVE and BELOW surfaces of adjacent faces are themselves adjacent.

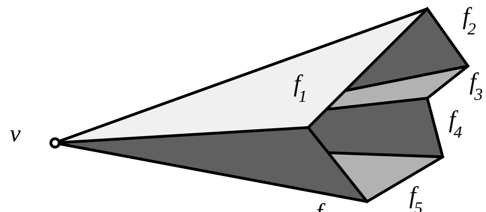

Fig. 9. A Cone

We will consider the arrangement of a polyhedron corner with *two* vertices, since this is a much more interesting situation and the one-vertex case follows trivially. The features of interest are two points, a and b, corresponding to the two vertices to be arranged, the point v which lies at the cone vertex, the planes p_i which correspond to the cone faces f_i, and the lines ℓ_i which correspond to the cone edges:

$$F = \{a, b, v, p_1, p_2, \ldots, p_n, \ell_1, \ell_2, \ldots, \ell_n\}.$$

Only a limited set of relations will be considered:

- $\textsc{location}(a, p) \in \{\textsc{below}, \textsc{on}, \textsc{above}\}$.
- $\textsc{incidence}(a, b) \in \{\textsc{on}, \textsc{off}\}$. A point a is ON a point b, or is not.
- $\textsc{incidence}(a, \ell) \in \{\textsc{on}, \textsc{off}\}$. A point a is ON a line ℓ, or is not.
- $\textsc{incidence}(\ell, p) \in \{\textsc{on}, \textsc{off}\}$. A line ℓ is embedded ON a plane p, or is not.

The set of allowable relations, R_{allow}, constrains v to lie ON each p_i, defines $\ell_i = p_i \cap p_{i+1}$, and permits the points a and b to have any relations whatsoever to the other features. Finally, the problem instance contains the topological constraint that a cone must be formed by the faces $f_1, f_2, \ldots, f_n$ which are defined by the planes $p_1, p_2, \ldots, p_n$.

$$\begin{aligned} R_{allow} = \{ \quad & v \text{ ON } p_i, \\ & \ell_i \text{ ON } p_i, \quad \ell_i \text{ ON } p_{i+1}, \\ & a \text{ BELOW } p_i, \quad a \text{ ON } p_i, \quad a \text{ ABOVE } p_i, \\ & b \text{ BELOW } p_i, \quad b \text{ ON } p_i, \quad b \text{ ABOVE } p_i, \\ & a \text{ ON } \ell_i, \quad b \text{ ON } \ell_i, \quad a \text{ OFF } \ell_i, \quad b \text{ OFF } \ell_i, \\ & a \text{ ON } v, \quad b \text{ ON } v, \quad a \text{ OFF } v, \quad b \text{ OFF } v, \\ & a \text{ ON } b, \quad a \text{ OFF } b \ \} \end{aligned}$$

Consider the nondegenerate case in which a and b cannot lie ON any feature. Then the only ambiguous relations in R are $\textsc{location}(a, p_i)$ and $\textsc{location}(b, p_i)$:

$$\begin{aligned} R_{nondeg} = \{ \quad & v \text{ ON } p_i, \\ & \ell_i \text{ ON } p_i, \quad \ell_i \text{ ON } p_{i+1}, \\ & a \text{ BELOW } p_i, \quad a \text{ ABOVE } p_i, \\ & b \text{ BELOW } p_i, \quad b \text{ ABOVE } p_i, \\ & a \text{ OFF } \ell_i, \quad b \text{ OFF } \ell_i, \\ & a \text{ OFF } v, \quad b \text{ OFF } v, \\ & a \text{ OFF } b \ \} \end{aligned}$$

The following theorem is proved elsewhere by the author.[36] The proof is rather long and is not included here. The question posed at the beginning of this section is answered by the corollary.

Theorem 4 *For any complete set of relations $R \subset R_{nondeg}$, there exist a cone and two points that satisfy R.*

Corollary 1 *A model exists for any nondegenerate arrangement of a cone and one point.*

Consider the degenerate case in which a and b can lie ON various features. Then R_{allow} is unchanged. Four new critical properties are introduced in Table 4, where a, b, and c are points, ℓ is a line, p and q are planes, and f_i is any feature of F.

Table 4. Critical Properties For Cone Arrangements

CP 9 :	a ON $b \;\Rightarrow\;$ LOCATION$(a,p) =$ LOCATION(b,p)
CP 10 :	a ON $f_1 \;\wedge\; a$ ON $f_2 \;\Leftrightarrow\; a$ ON $f_1 \cap f_2$
CP 11 :	Let a, b, c ON ℓ, c ON p, q, and ℓ OFF p, q. Then LOCATION$(a,p) =$ LOCATION$(b,p) \;\Leftrightarrow\;$ LOCATION$(a,q) =$ LOCATION(b,q)
CP 12 :	a ON $f_1 \;\wedge\; a$ ON $f_2 \;\Rightarrow\; f_1 \cap f_2 \in F$

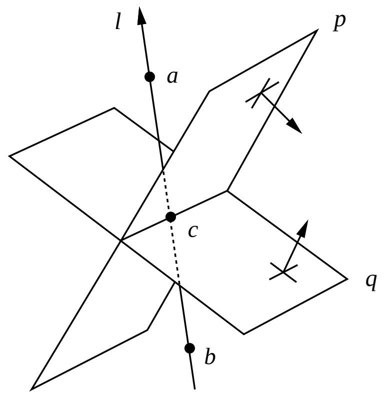

Fig. 10. Example of CP 11

CP 9 states the coincident points have the same LOCATION relation to every plane. CP 10 states that a point lies ON two features exactly when it lies ON their intersection. CP 11 ensures that points that share a line have consistent LOCATION relations with planes that intersect that line (see Figure 10). CP 12 states that no point can lie ON two other features unless their intersection is also a feature in F. This last rule constrains the set of relations $R \subseteq R_{allow}$ such that each of the points a and b has one of the following relations to the planes of F:

- the point is not ON any plane,
- the point is ON exactly one plane, p_i,
- the point is ON exactly two adjacent planes, p_i and p_{i+1}, or
- the point is ON every plane.

The following theorem is proved elsewhere by the author.[36] The proof is even more tedious than the proof of Theorem 4 and is not presented here. Note that Theorem 5 states sufficient, but not necessary, conditions for the existence of a model, since CP 12 precludes some arrangements that have models.

Theorem 5 *Let F and R_{allow} be as defined above for the arrangement of a corner and two vertices. Let $R \subset R_{allow}$ be a complete set of relations among the features of F. Then R has a model if R obeys CP 9 through CP 12.*

A simple extension to Theorem 5 shows that these properties suffice for arrangements of two polyhedron corners with each other, since the faces of one corner can be positioned independently of the faces of the other corner.

Corollary 2 *A model exists for any arrangement of two polyhedron corners that obeys CP 9 through CP 12, as long as only* LOCATION *and* INCIDENCE *relations are considered.*

5. A Robust Polyhedral Intersection Algorithm

The critical properties of the preceding sections were incorporated in a polyhedral intersection algorithm. The algorithm was extensively tested and never failed to produce a valid polyhedron of intersection.

At an abstract level, the author's polyhedral intersection algorithm operates as shown in Figure 11. The raw input to the algorithm consists of symbolic data describing the topology of the two polyhedra to be intersected, and numeric data describing the approximate location of each feature of the polyhedra (using a region of tolerance for each feature).

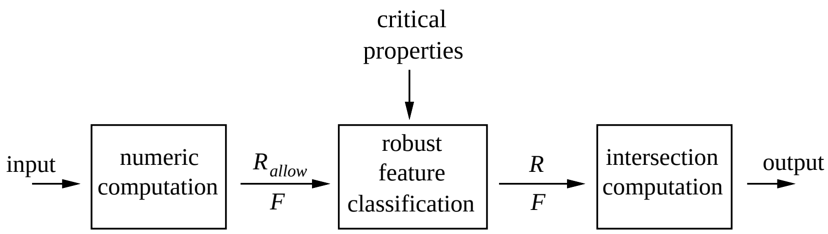

Fig. 11. Abstract Intersection Algorithm

From this raw input, the "numeric computation" component computes a set F of features and a set R_{allow} of allowable relations among these features. If two features have nonintersecting regions of tolerance then their relation is unambiguous and R_{allow} contains exactly one allowable relation for these features. If the relation between two features is ambiguous then R_{allow} contains all possible relations for these features.

The "robust feature classification" component takes F and R_{allow}, along with the critical properties derived in Section 4, and produces a complete set of relations $R \subseteq R_{allow}$ that satisfies the critical properties. CP 11 was not included in the critical properties since it can be replaced by CP 5 through CP 8 when the ORDER relation is considered (this relation was not considered in Theorem 4 and Theorem 5, which used CP 11).

Finally, the "intersection computation" takes the purely symbolic data of the previous component and computes the topology of the polyhedron of intersection. The relevant approximate plane equations taken from the raw input.

5.1. Intersection Computation

The intersection computation will only be discussed briefly since it is not the focus of this work: it serves only as a testbed for the critical properties. For details of various intersection algorithms, the reader is directed to any one of Karasick,[28] Vanecek,[37] Segal and Séquin,[27] or Laidlaw, Trumbore, and Hughes.[38] Chapter 4 of Karasick's dissertation[28] also provides a good survey of other intersection algorithms.

The author's intersection algorithm compares every edge of one polyhedron with every face of the other polyhedron, and vice versa. If the endpoints of the edge span the plane of the face, then the intersection point is considered: if it is on the interior of the face then it is tagged as a vertex of the polyhedron of intersection. Due to the number of special cases which can arise in intersecting an edge and a face, the bulk of the algorithm lies in this step. Once the vertices have been classified the resulting data structure is traversed to compute the topology of the polyhedron of intersection.

5.2. Robust Feature Classification

Robust feature classification is achieved with a constraint propagation algorithm. In this section the algorithm is discussed briefly. For the details of such algorithms, the reader is directed to Waltz,[39] Huffman,[40] or Clowes.[41]

The input to the algorithm consists of a set F of features, a set R_{allow} of allowable relations among these features, and a set of critical properties. The output is a complete set of relations $R \subseteq R_{allow}$ that satisfies the critical properties.

R_{allow} might contain several alternative labelings for a given relation. The constraint propagation algorithm chooses one of the alternatives (that is, the relation is instantiated), removes the other alternatives from R_{allow}, and propagates the consequences of this instantiation by evaluating the critical properties that depend upon the instantiated relation (this can be efficiently implemented with pointers between the data structures for the relations, the features, and the critical properties). If the antecedent of a critical property becomes true then the relation that is its consequent is instantiated, perhaps triggering the evaluation of even more critical properties. When no more critical properties remain to be evaluated, another relation is chosen for instantiation. This procedure is iterated until no alternative labelings remain in R_{allow}.

Constraint propagation has exponential time complexity because the algorithm might make an inconsistent choice and have to backtrack when this inconsistency is discovered. However, backtracking never occured in any of the tests performed on the algorithm. This is possibly due to the following aspects of the algorithm:

- Only LOCATION relations were chosen for instantiation. All other types of relation were only instantiated as a consequence of a deduction from one of the critical properties.

- The critical properties were augmented with logical equivalents. For example, the property $A \wedge B \Rightarrow C$ would be augmented with the properties $A \wedge \overline{C} \Rightarrow \overline{B}$ and $B \wedge \overline{C} \Rightarrow \overline{A}$.

The advantage of the constraint propagation approach is that the intersection algorithm runs at normal speed when presented with unambiguous input and only slows down when it is necessary to make deductions about ambiguous input.

5.3. Testing a Polyhedral Intersection Algorithm

A standard test to measure the robustness of an intersection algorithm is to intersect two identical objects, usually cubes.[28,38] Both objects are centered at the origin, but one is rotated by a small angle through an arbitrary axis. This is called the *small angle test.*

With respect to this axis, the two objects, and the precision of the floating point arithmetic, there are two distinguished angles: α and β. Below the angle α all corresponding (approximate) features of the two objects are indistinguishable with numeric tests. Above the angle β all relations between features can be determined with numeric tests, including those relations involving features derived by combining the two objects (for example, points defined as the intersection of an edge and a face).

Any nonrobust intersection algorithm that uses the principle of "if it's close to zero, then it is zero" (described in Section 1.2) should successfully output the original object for angles below α, since it would determine that the two objects are identical. Such an algorithm should also be successful for angles above β, since for these angles the numeric tests would be reliable. It is in the *region of instability* $[\alpha, \beta]$ that nonrobust algorithms fail. They fail by making inconsistent decisions about relations that cannot be determined numerically.

Another standard test is to intersect an object with a rotated version of itself, then intersect the resulting solid with a rotated version of *itself*, an so on.[28] Each iteration approximately doubles the number of faces. This is called the *repeated intersection test.*

The repeated intersection test results in solids that have vertices of high degree, so failures induced by this test occur in more complicated situations than those that occur in the small angle test with cubes. However, these complicated situations can be produced by the small angle test with objects other than cubes.

For example, the author has observed that the small angle test with octahedra (having vertices of degree four) results in very complicated intersections when two vertices of different octahedra are close. In such situations, the instantiation of a relation involving a vertex on one side of the octahedron can affect the relations involving a vertex on the opposite side of the octahedron. This long distance effect of a local instantiation is successfully handled by local robustness and constraint propagation.

5.4. Testing the Locally Robust Intersection Algorithm

Two small angle tests were performed on the author's locally robust intersection algorithm: one involving two cubes, the other involving two octahedra. The repeated intersection test was not performed because the intersection algorithm was extremely slow and memory intensive (an attempt at this test on a Symbolics Lisp Machine caused the machine to run out of memory after several iterations). However, the small angle test with octahedra produces the same types of situations that cause failure in the repeated intersection test, including long distance effects such as that described above.

For the small angle test, the two simple procedures shown in Figure 12 can determine α and β. In these procedures, $\text{OBJ}(\theta, v)$ is the object rotated by angle θ about the axis v, and ε is the machine precision.

```
Find-alpha( OBJ, v )

    β ← 10.0 degrees
    α ← 0.0 degrees
    while |β − α| > ε do
        θ ← (α + β)/2
        Intersect OBJ(0.0, v) and OBJ(θ, v).  If all corresponding
        features are numerically indistinguishable, then
            α ← θ
        else
            β ← θ
    Return α

Find-beta( OBJ, v )

    β ← 10.0 degrees
    α ← 0.0 degrees
    while |β − α| > ε do
        θ ← (α + β)/2
        Intersect OBJ(0.0, v) and OBJ(θ, v).  If all numeric
        tests are unambiguous, then
            β ← θ
        else
            α ← θ
    Return β
```

Fig. 12. Determining α and β

Some researchers suggest that their intersection algorithms are robust because the algorithms successfully compute α with the procedure of Figure 12, or because the algorithms fail during this computation when the distance between α and β is very small. While the procedures of Figure 12 can sometimes induce failure in a nonrobust algorithm, completing either one of them successfully does not mean that the intersection algorithm is robust: it could be that the angles encountered in the search fortuitously avoided inconsistent decisions by the algorithm.

Since we cannot hope to test every representable angle between α and β, the best approach is to test a large sample of them. In doing so, we hope to test many different combinations of ambiguous relations between features. Failure is more likely near the lower angle α, since there are more ambiguous relations near this angle. Thus, more tests should be performed near α.

The author's polyhedral intersection algorithm was tested for a pair of unit cubes and a similar-sized pair of regular octahedra. The octahedron test is significant because in the early stages of the algorithm's design, it would cause the algorithm to fail where the cube test could not. Thus algorithms that pass the small angle test for cubes may, in fact, not be robust.

The algorithm was tested on a Symbolics 3675 Lisp Machine using twenty-three bit floating point arithmetic. The features of the cube and octahedron were represented with an accuracy of 10^{-5}, slightly less accurate than the machine epsilon of $2^{-23}(\approx 10^{-7})$. Tables 5 and 6 summarize the tests. Each table includes the following information:

- the number of vertices, edges, and faces in the resulting polyedron of intersection,
- the number of coincident vertices between the two rotated objects,
- the number of ambiguous numeric relations that were arbitrarily instantiated by the robust feature classification component of the algorithm,
- the number of ambiguous numeric relations that were deduced from the critical properties, and which were of interest to the intersection computation component of the algorithm (others relations were deduced, but were not of direct interest), and
- the total number of ambiguous numeric relations.

The algorithm was also tested by removing the critical properties from the robust feature classification component and using in their place the principle of "if it's close to zero, then it is zero." As expected, this nonrobust version of the algorithm failed at many of the angles for which the robust version did not fail. This suggests that a locally robust algorithm avoids those types of failure that occur in practice.

5.4.1. The Cube Test

The axis of rotation for the cube test was the vector $[1,2,3]$. This was chosen to avoid rotating through any of the vertices or edges of the cube, in the hope that more interesting ambiguities would arise. A few tests were performed with an axis of $[1,1,1]$ passing through two opposite vertices of the cube, but fewer ambiguities arose in these tests than in those with the axis of $[1,2,3]$. The computed α and β for the cube were 0.00066126644 and 0.96176076 degrees, respectively. The algorithm succeeded on all angles that were tested. From over one hundred such tests, a representative sample of 25 are summarized in Table 5.

Note that at angle 0.9617608 (shown at the bottom of Table 5) there are still 90 ambiguous relations, which appears to contradict the definition of β, since 0.9617608

Table 5. Data From the Cube Test

angle (degrees)	*result vertices*	*result edges*	*result faces*	*coincident vertices*	*relations instantiated*	*relations deduced*	*relations total*
0.0006612664	8	12	6	8	7	97	1200
0.00066126644	8	12	6	2	9	93	1349
0.0007324219	8	14	8	1	10	90	1307
0.0009765625	10	16	8	0	9	65	1820
0.0015415343	14	24	12	0	8	30	1969
0.0021935818	20	30	12	0	6	14	2172
0.00390625	20	30	12	0	6	12	1814
0.00443137	18	28	12	0	2	6	1225
0.0078125	18	28	12	0	2	14	1143
0.015625	18	28	12	0	2	14	924
0.03125	20	30	12	0	0	16	716
0.0625	20	30	12	0	0	16	624
0.125	20	30	12	0	0	16	624
0.13524757	20	30	12	0	0	16	624
0.25	20	30	12	0	0	16	624
0.3655514	20	30	12	0	0	16	578
0.47362018	20	30	12	0	0	8	453
0.5	20	30	12	0	0	8	443
0.52384853	20	30	12	0	0	8	443
0.6977829	20	30	12	0	0	8	439
0.750525	20	30	12	0	0	8	439
0.93798125	20	30	12	0	0	8	421
0.9617593	20	30	12	0	0	4	355
0.96176076	20	30	12	0	0	2	222
0.9617608	20	30	12	0	0	0	90

Table 6. Data From the Octahedron Test

angle (degrees)	*result vertices*	*result edges*	*result faces*	*coincident vertices*	*relations instantiated*	*relations deduced*	*relations total*
0.0016037918	6	12	8	6	6	96	1638
0.001603792	6	12	8	1	6	92	2107
0.0016495044	6	12	8	1	6	89	2101
0.001953125	7	13	8	1	7	75	2354
0.0027060139	11	18	9	0	9	49	3127
0.00390625	13	21	10	0	8	44	3495
0.0042598094	15	24	11	0	7	32	3408
0.004904998	21	32	13	0	8	31	3813
0.0070919236	21	32	13	0	7	14	2898
0.008351261	21	32	13	0	6	10	2646
0.01	24	36	14	0	5	3	2775
0.018272676	24	37	15	0	4	8	1672
0.033572037	24	37	15	0	4	8	1585
0.058525786	25	38	15	0	3	5	1401
0.061003245	25	38	15	0	3	5	1401
0.08395079	26	39	15	0	2	2	1238
0.125	27	41	16	0	1	19	1166
0.19195558	28	42	16	0	0	20	972
0.5	28	42	16	0	0	20	862
1.0	28	42	16	0	0	12	587
1.6489155	28	42	16	0	0	8	408
2.3067856	28	42	16	0	0	4	235
2.505	28	42	16	0	0	4	737
3.481383	28	42	16	0	0	1	194
3.4813833	28	42	16	0	0	0	217

is greater than β. However, these unknown relations were not of interest to the algorithm.

5.4.2. The Octahedron Test

The axis of rotation for the octahedron test was the vector $[1, 2, 3]$. The computed α and β for the octahedron were 0.001603792 and 3.481383 degrees, respectively. The algorithm succeeded on all angles that were tested. From over one hundred such tests, a representative sample of 25 are summarized in Table 6.

6. Conclusion

The philosophy of this work is to build practical, reliable algorithms on a firm, theoretical foundation. Local robustness provides this foundation, and the polyhedral intersection algorithm of Section 5 demonstrates that local robustness really does work in practice. The contributions of this paper are the following:

- a new theory of local robustness that bridges the gap between the theory and the practice of building robust algorithms,
- a set of critical properties that characterizes locally robust polyhedral arrangements, and
- an intersection algorithm which demonstrates that local robustness works in practice.

We should not spend so much energy on developing perfectly robust, perfectly efficient algorithms. Nor should we ignore the vagaries of floating point arithmetic, and hope that our "real arithmetic" algorithms will work when implemented. Local robustness provides the middle ground with which to build practical, reliable algorithms on a firm, theoretical foundation.

6.1. Future Work

Local robustness should be applied to other geometric problems, such as building Voronoi diagrams and finite-element meshes (however, for line arrangements the approach of Milenkovic[6] is probably better, since it has already been proven robust and has been efficiently implemented).

A better, locally robust polyhedral intersection algorithm should be developed. Instead of incorporating the robustness rules in an explicit database (as was done with the constraint propagation approach) a better algorithm would incorporate them implicitly and efficiently.

Local robustness should be applied to more aspects of solid modeling, not just to polyhedral intersection. The most important aspect would be a locally robust *rounding algorithm*: since coordinate transformations on a solid (such as translation, rotation, or scaling) generally increase the length of the floating point numbers in the computer representation of the solid, these floating point numbers must be periodically rounded while maintaining the same solid topology. Since this rounding problem is NP-hard,[9] it seems to be an ideal application of local robustness. The robustness rules presented in Section 4 would probably work for such a rounding algorithm. Another aspect of solid modeling is that of nonpolyhedral solids: a characterization of locally robust intersection algorithms for various classes of these solids should be developed.

7. Acknowledgements

I would like to thank Chanderjit Bajaj, Paul Chew, Jim Cremer, Christoph Hoffmann, John Hopcroft, Victor Milenkovic, and Kjartan Stefansson for the many interesting discusssions that we have had over the course of this work. Thanks also to the reviewers for their helpful suggestions.

This research has been supported by Nasa Training Grant NGT-50327, Office of Naval Research Contract N00014-88K-0591, Office of Naval Research Grant N00014-89J-1946, and National Science Foundation Grant IRI-9006137.

8. References

1. L. P. Chew. Guaranteed-quality triangular meshes. Technical Report 84-602, Computer Science Department, Cornell University, 1989.
2. F. P. Preparata and M. I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, 1985.
3. C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *ACM Annual Symposium on Computational Geometry*, pages 106–117, 1988.
4. C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *Computer*, 22(3):31–41, 1989.
5. C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann, 1989.
6. V. J. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic.* PhD thesis, Carnegie Mellon University, 1988.
7. D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 143–152, 1986.
8. K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. *Journal of Information Processing*, 12(4):380–393, 1989.
9. V. J. Milenkovic and L. R. Nackman. Finding compact coordinate representations for polygons and polyhedra. In *ACM Annual Symposium on Computational Geometry*, pages 244–252, 1990.
10. T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In *ACM Annual Symposium on Computational Geometry*, pages 119–125, 1987.
11. J. E. Hopcroft and P. J. Kahn. A paradigm for robust geometric algorithms. Technical report 89-1044, Computer Science Department, Cornell University, 1989.
12. V. J. Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, 37:377–401, 1988.
13. V. J. Milenkovic. Double precision geometry: A general technique for calculating line and segment intersections using rounded arithmetic. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 500–505, 1989.

14. V. J. Milenkovic. Calculating approximate curve arrangements using rounded arithmetic. In *ACM Annual Symposium on Computational Geometry*, pages 197–207, 1989.

15. S. Fortune. Stable maintenance of point set triangulations in two dimensions. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 494–499, 1989.

16. D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: Building robust algorithms from imprecise calculations. In *ACM Annual Symposium on Computational Geometry*, pages 208–217, 1989.

17. D. Salesin. *Epsilon Geometry: Building Robust Algorithms From Imprecise Computations*. PhD thesis, Stanford University, 1991.

18. Z. Li and V. J. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *ACM Annual Symposium on Computational Geometry*, pages 234–243, 1990.

19. K. Sugihara and M. Iri. Construction of the Voronoi diagram for over 10^5 generators in single-precision arithmetic. In *Abstracts of the First Canadian Conference on Computational Geometry*, page 42, 1989.

20. K. Sugihara and M. Iri. Two design principles of geometric algorithms in finite-precision arithmetic. *Appl. Math. Lett.*, 2(2):203–206, 1989.

21. K. Sugihara. A robust intersection algorithm based on Delaunay triangulation. Technical report 91-011, Department of Computer Sciences, Purdue University, 1991.

22. K. Sugihara, Y. Ooishi, and T. Imai. Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms. In *Canadian Conference on Computational Geometry*, pages 36–39, 1990.

23. B. Brüderlin. Detecting ambiguities: An optimistic approach to robustness problems in computational geometry. Technical report UUCS-90-003, Computer Science Department, University of Utah, 1990.

24. B. Brüderlin. Robust regularized set operations on polyhedra. In *Hawaii International Conference on System Sciences*, pages 691–700, 1991.

25. S. Fang and B. Brüderlin. Robustness in geometric modeling – tolerance-based methods. In *Lecture Notes in Computer Science 553: "Proceedings of the International Workshop on Computational Geometry CG '91"*, pages 85–101. Springer-Verlag, 1991.

26. S. Fang and B. Brüderlin. Robust geometric modeling with implicit surfaces. In *International Conference on Manufacturing Automation*, 1992. to appear.

27. M. Segal and C. Sequin. Partitioning polyhedral objects into nonintersecting parts. *IEEE Computer Graphics and Applications*, 8:53–67, January 1988.

28. M. Karasick. *On the Representation and Manipulations of Rigid Solids*. PhD thesis, McGill University, 1989.

29. C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, 1989.

30. M. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics (SIGGRAPH Procedings)*, 24(4):105–114, 1990.

31. M. Segal and C. Séquin. Consistent calculations for solid modeling. In *ACM Annual Symposium on Computational Geometry*, pages 29–37, 1985.

32. D. Dobkin and D. Silver. Recipes for geometry & numerical analysis – Part I: An

empirical study. In *ACM Annual Symposium on Computational Geometry*, pages 93–105, 1988.

33. C. M. Eastman and K. Preiss. A review of solid shape modelling based on integrity verification. *Computer-Aided Design*, 16(2):66–80, 1984.
34. H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. In *ACM Annual Symposium on Computational Geometry*, pages 118–133, 1988.
35. C. Yap. A geometric consistency theorem for a symbolic perturbation scheme. In *ACM Annual Symposium on Computational Geometry*, pages 134–142, 1988.
36. A. J. Stewart. *The Theory and Practice of Robust Geometric Computation, or, How to Build Robust Solid Modelers.* PhD thesis, Cornell University, 1991. Available as Cornell technical report TR 91-1229.
37. G. Vanêĉek, Jr. Obtaining boundaries with respect: A simple approach to performing set operations on polyhedra. Technical report CSD-TR-920, Department of Computer Sciences, Purdue University, 1989.
38. D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes. Constructive solid geometry for polyhedral objects. *Computer Graphics (SIGGRAPH Procedings)*, 20(4):161–170, 1986.
39. D. Waltz. Generating semantic descriptions from drawings of scenes with shadows. In P. Winston, editor, *The Psychology Of Computer Vision*, pages 19–92. McGraw-Hill, 1975.
40. D. Huffman. Impossible objects as nonsense sentences. In R. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 295–323. Elsevier, 1971.
41. M. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.