# Towards Implementing Robust Geometric Computations

Christoph M. Hoffmann[1]
Computer Science Department
Purdue University

John E. Hopcroft[2]
Computer Science Department
Cornell University

Michael S. Karasick[3]
Computer Science Department
Cornell and McGill Universities

## 1. Introduction

Computational geometry has the unique opportunity to bridge the sharp gap between theoretical and applied computer science. Indeed, practical computations with geometric objects are of intense interest to a wide range of applied work including computer aided design, robotics, mathematics, engineering, etc. At the same time, these computations pose many challenging problems of considerable theoretical depth and interest.

Implementing numerically robust algorithms for computational geometry is a nontrivial task. Except for very limited classes of geometric objects, it is incorrect to assume that infinite precision arithmetic or symbolic computation will yield correct implementations, because basic operations such as translation or rotation introduce inaccuracies into the representation. For example, a boundary representation of a polyhedral solid consists of two components: A topological component describing the incidence of vertices, edges and faces, and a numerical component consisting of face equations. When the coefficients of the face equations have been truncated, the topology may claim that four faces meet at a vertex when in fact the face equations indicate that they meet in a structure consisting of two vertices connected by a very short edge. This inconsistency can lead to a fatal error in a program that is manipulating the representation and is relying on its consistency for program correctness.

It is desirable to assume that the incidence relations are correct and that the numeric data are only approximations to the real data. For instance, [10] shows that the number of significant digits more than quintuples when intersecting linear, three-dimensional structures. Moreover, rotating a line by exact angles such as $\sin(\pi/7)$ requires the symbolic representation of high degree algebraic numbers. In these and other cases, the machinery implementing exact arithmetic operations soon dominates the running time of an algorithm and renders it useless in practice.

It is clear that infinite precision computations cannot deal with inaccuracies of the numerical data: Typically, an algorithm computes a numerical quantity, say $x$, and then derives logical information by testing whether $x$ is less than, equal to, or greater than zero. It is at this point where there is potential for trouble: When $x$ is less than a certain threshold $\varepsilon$, the numerical inaccuracies of the input and, possibly, the arithmetic computations simply yield no further information. Arbitrarily assuming that $x = 0$ leads to program failure. Assuming that the input is correct as written yields, at best, an unpleasant proliferation of microscopically small geometric structures, but may also lead to contradictory information and program failure.

In this paper, we discuss several paradigms for developing provably correct implementations of geometric algorithms, accounting for the possibility of imprecise numerical input data. These paradigms are based on the concept that, in the presence of numerical uncertainty, the logical decision cannot be based on the arithmetic computation alone, but must be consistent with all previous such decisions. It is our experience that even in situations where a full correctness proof of the algorithm is not yet completed, this paradigm leads to robust *and* efficient implementations [5,6]. We illustrate these ideas in a variety of intersection problems.

## 2. The Reasoning Paradigm

If we base logical tests such as incidence on numerical calculation, assuming approximate data and arithmetic operations of limited precision, then there is an interval of uncertainty in which the numerical data cannot yield further information. In such a situation, a decision must be made that has to be consistent with other such decisions and with the topological data. For example, points that have been declared collinear by the topology must be treated as collinear points by the algorithm. Making decisions consistently requires symbolic reasoning, and it is important to understand how complex the reasoning steps could be.

Let $M$ denote a geometric object such as a polygon and

---

[1] Supported in part by NSF grants DCR 85-12443 and CCR 86-19817 and ONR contract N0014-86-K-0465
[2] Supported in part by NSF grants DCR 85-02568 and DMC 86-17355 and ONR contract N0014-86-K-0281
[3] Author's Address: IBM T.J. Watson Research Center, Yorktown Heights, NY.

let $R$ denote a representation of the object. The difference between an object and its representation is that the object can have equations with arbitrary real numbers whereas the equations in the representations are fixed precision numbers. A representation has associated with it a set of models. A *model* is a geometric object with the same incidence structure as the representation and numeric specifications that approximate those of the representation. For many geometric objects the representation is a model of itself, called the *natural* model. A binary operation such as intersection is said to be *correct* for input representations $R_1$ and $R_2$ if it produces an output representation $R_3$ such that there exist models $M_1$, $M_2$, and $M_3$ where $M_i$ is a model of $R_i$ and $M_3 = M_1 \cap M_2$.

The fact that the algorithm is correct in this sense does not mean that it can be used naively as a subroutine in a larger problem. The notion of correctness is one which applies only to a single operation. To see this, consider the problem of intersecting robustly a pair of line segments. Each line segment is represented by a pair of points whose coordinates are only approximately correct. In our framework, a correct implementation can be given using exact or approximate computation. The algorithm will give correct answers for line segment intersection, but does not account for possible additional topological structure. Therefore, it cannot be used unaltered to implement polygon intersection, since the property of consecutive edge incidence in a common vertex is not accounted for in the computation.

We examine the utility of the reasoning paradigm when intersecting two and three polygons, and discuss the complexity of the needed reasoning steps. As we shall see, virtually no reasoning is required when intersecting two polygons, provided the algorithm is based on vertex/vertex and vertex/edge incidence computations. This is not the case for simultaneously intersecting three polygons. There, theorems from projective geometry must be accounted for.

## 3. Intersecting Two Polygons

A *representation* for polygons consists of the following data:

(1) Symbolic vertex specifications, of the form $v = (l, l')$, where $l$ and $l'$ are lines.

(2) Symbolic edge specifications, of the form $e = (v, w)$ where $v$ and $w$ are vertices.

(3) Numeric line specifications of the form $l = ax + by + c$, where $a$, $b$, and $c$ are numbers, e.g., in floating-point format. Here line equations are *oriented* such that the gradient $(a,b)$ points to the polygon exterior along the edge.

Note that the natural model polygon may not be simple. We quantify the accuracy between a representation and a model by

*Definition.* A representation $R$ is *ε-correct* provided there is a model polygon $M$ that satisfies the symbolic information of the representation, is a simple polygon, and its vertices are within $\varepsilon$ of the vertices of the natural model.

Next, we need the concept of *minimum feature separation.* Intuitively, a representation has minimum feature separation if no two vertices are closer than a certain tolerance, all edges are larger than a certain minimum length, and consecutive edges have angles not smaller or larger than specific critical values. The purpose of this definition is to limit the effect that perturbing the numerical data has on the polygon geometry. The precise statement is the following:

*Definition.* A representation $A$ has minimum feature separation if consecutive edges form an angle larger than $\alpha$ and smaller than $\pi - \alpha$, if all edges are longer than $3\,\varepsilon$, two vertices are separated by at least $3\,\varepsilon$, and no vertex is closer to an edge than $3\,\varepsilon$.

Here $\varepsilon$ is a function of $\alpha$ and represents the maximum error the determination of vertex coordinates can incur assuming that the lines intersecting in the vertex are at an angle $\alpha$. For example, the *condition number* [3] of the two line gradients can be used to define $\varepsilon$.

Suppose a vertex of one model lies on an edge of the other model. Then the vertex and the edge are said to be constrained. A vertex so constrained in turn constrains its adjacent edges. Thus, an edge can be constrained by its own vertices as well as by vertices of the other object. An edge with more than two constraints is *overconstrained.*

*Lemma 1.* Let $M_1$ and $M_2$ be two model polygons. Then not every edge of $M_1$ and every edge of $M_2$ can be overconstrained.

*Corollary.* There is at least one edge of $M_1$ or $M_2$ that is not overconstrained.

*Lemma 2.* Let $R_1$ and $R_2$ be two representations with a set of incidence constraints of the forms "vertex $u$ is on edge $e$," and "vertex $v$ and $w$ coincide." Then there are models of $M_1$ and $M_2$ such that the incidence constraints are satisfied provided there is at least one edge that is not overconstrained.

Intuitively, the proof of Lemma 2 works as follows: Remove all edges that are not overconstrained and also remove their end points. By a counting argument, there remain edges that now are not overconstrained. These are removed, along with their end points. This process continues with the remaining edges until all edges are removed. The edges are now placed in reverse order of removal.

We can obtain an intersection algorithm based on Lemma 2 as follows: Here $\varepsilon$ depends on the minimum feature separation constant and the norm of the line equation $L$.

(1) Say that a vertex $u$ is on an edge $e = (v, w)$ if $L(v) < \varepsilon$, where $\varepsilon$ is a chosen tolerance and $L$ is the line equation for $e$, and if $u$ is between $v$ and $w$ and not close to either vertex.

(2) Say that vertices $u$ and $v$ are coincident if $u$ is close to $v$.

It is possible that the algorithm overconstrains every edge of both polygons. A case for potential trouble is shown in Figure 1. This case is excluded by minimum feature separation. A more subtle difficulty arises as shown in Figure 2 where the tests announce incidences $B$ on $DE$ and $E$ on $BC$ implying $B = E$ or $DE$ and $BC$ are collinear. The test whether two vertices are near must be such that if $u$ and $v$ are not coincident, then neither $u$ nor $v$ is on both edges defining the other vertex.

*Theorem 1.* Let $R_1$ and $R_2$ be two representations with $\varepsilon$ correct models. Then there exists a representation $R_1 \cap R_2$ with a model $M_3$ such that there are models $M_1$ and $M_2$ of $R_1$ and $R_2$ with $M_3 = M_1 \cap M_2$. Moreover, there exists a $\delta$ such that all models are $\delta$-correct.

Note that the theorem shows correctness and quantifies the accuracy of the intersection algorithm. The accuracy crucially depends on the incidence tests, especially the vertex/vertex incidence tests.

After two representations have been intersected, the result need not satisfy the minimum feature separation condition for $\varepsilon$. Thus, a post-processor may be needed to restore the minimum feature separation condition. This may require the obliteration of short edges, i.e., affects the symbolic data as well as the numeric data of the representation. As noted in [7,11], adjusting the numeric data to fixed precision rational data is expensive. It is not difficult to extend these results on intersecting polygons to embedded planar graphs, provided that no relationships of collinearity or parallelism are assumed among the edges.

We can now explain why an algorithm for intersecting polygons based on vertex incidence tests is robust whereas one based on edge intersection computation is not. All vertex-on-edge questions are independent but the set of edge intersection questions is not. Asking if a vertex is on the infinite line defined by an edge is not allowed. The reason for this is that these questions add additional constraints on edges and destroy the independence argument. In Figure 3, edges AB and CD do not intersect and a vertex can be close to at most one of the edges. However, asking if vertex $v$ is on the infinite line defined by AB and on the infinite line defined by CD, could result in a constraint on both edges. In fact, a vertex could constrain an arbitrarily large number of edges and the proof of Lemma 2 would not work. Similarly, we must require that the polygons to be intersected be simple. If edge AB were to cross edge CD and vertex $v$ were close to the point of intersection, then it would again constrain two or more edges.

Even though there are no relationships assumed to hold among the edges of each input polygon, edges in the output polygon may have such relationships. For example, in Figure 4 two sides of a polygon must be on the same infinite line. This will cause a problem when we try to intersect the result with a third polygon. We may choose to discard all such relationships. Then we can iterate polygon intersection. However, in that case the algorithm cannot be used as a subroutine by a more general algorithm whose correctness depends on some global property that might be destroyed. One also should be aware of the fact that the pairwise intersection algorithm is not associative. In general, $(R_1 \cap R_2) \cap R_3 \neq R_1 \cap (R_2 \cap R_3)$. This suggests that there should be two definitions ror correctness of the polygon intersection algorithm: one definition for the isolated problem of intersecting two polygons and another definition if the intersection algorithm is a subroutine of a larger computation. This is exactly analogous to the edge intersection problem.

## 4. Simultaneously Intersecting Three Polygons

Rather than intersecting polygons successively, we may consider intersecting more than two polygons simultaneously. We show that doing so introduces new complexities into the reasoning done to resolve numerical uncertainty.

When intersecting three polygons simultaneously, one cannot arbitrarily place a vertex with respect to a nearby edge as illustrated in Figure 5. Assume that we are given three polygons $X$, $Y$ and $Z$, whose boundaries include the line segments shown in Figure 5. If one claims the incidences

$$(A,A'),\ (C,C'),\ (1,1'),\ (2,2'),\ (3,3'),\ (4,4'),\ (5,5'),\ \text{and}\ (6,6'),$$

then, by Pascal's Theorem, the edges (3,4), (1′,6), and $(A,C)$ must intersect in a common point:

*Pascal's Theorem.* If alternate vertices (1,3,5, and 2,4,6) of a hexagon are collinear then the three points that are the intersection of the lines (1,2) and (4,5), (2,3) and (5,6), and (3,4) and (6,1), are collinear.

The theorem is illustrated in Figure 6. Thus the problem of intersecting three polygons is sufficiently complex that determining if vertices are on edges requires a theorem prover powerful enough to handle theorems from projective geometry such as Pascal's Theorem. It is not difficult to prove that intersecting two embedded planar graphs with collinearity constraints requires proving all theorems of linear projective geometry ($\mathbf{P}^2$).

## 5. Line Sweep Algorithms

We consider the line segment intersection problem again as vehicle to explore other paradigms for implementing geometric computations: Given $n$ line segments $l_1, l_2, \ldots, l_n$ and a collection of subsets of the $l_i$ that appear to intersect at various points, find a consistent set of intersections.

Since the geometric structure of the problem is simple, the following solution could be proposed: Assume the natural model and compute all intersections with sufficient precision to find the exact intersection points. If the line coefficients are integers of length $L$, then a precision of $3L + 2$ is needed [10]. This approach is the *exact-as-written* paradigm. However, the coefficients in the line equations often are not exact, and it is unlikely that any three lines will intersect in a single point. In many applications close coincidence really would be coincidences were it not for the approximate line coefficients. In those cases it is desirable that we perturb the line positions so as to enlarge the number of common intersections.

Assume then that the equations of the lines are only approximate and adjust the equations so as to change a maximal number of near incidences of three lines to true incidences. This can be done as follows. Select a maximal set of lines with the property that no three lines go through any one point. These lines are said to be of type 1. The intersection point of a line of type $a$ with a line of type $b$ is said to be of type $a$-$b$. Each line not in $S$ appears to go through a type 1-1 intersection point. If a line not in $S$ appears to go through two or more type 1-1 intersection points, then add it to $S$ and call it type 2. New intersections of types 1-2 and 2-2 may be created. Now add to $S$ a maximal set of lines that go through type 1-1 intersection points and no other intersection points. These lines are designated type 3. All remaining lines appear to go through a type 1-1 intersection point and a point of type 1-2, 1-3, 2-2, 2-3 or 3-3. These remaining lines are designated type 4.

The equation for each line of type 1 is assumed to have exact coefficients. Coefficients of lines of type 2 are adjusted so that they go exactly through two points of type 1-1. Thus their coefficients require higher precision than the coefficients of type 1 lines. In turn lines of types 3 and 4 have their coefficients adjusted. Finite precision arithmetic is then used to test all other intersections. For example a line of type 2 may go through three intersection points of type 1 but only two of the points were used in defining it. The third point must be tested to determine if indeed it is a real intersection. In this manner we can insure that the set of answers for line intersection is indeed consistent. Again, with input coefficients of length $L$, a precision of $mL$ digits suffices, where $m$ is approximately 27, see [10]. Note, that implementing this strategy using the line sweep paradigm entails reporting the true intersection points off-line. A greedy on-line algorithm implementation would create lines of higher type and lead to an unacceptable growth in the number of digits required to test incidence correctly that is not independent of the problem size.

Although logically consistent, the model so obtained may require large coefficient perturbations. Figure 7 illustrates the problem: If we select lines $a$, $b$, $c$, and $d$ as a maximal set of type 1 lines, then a small perturbation of the input coefficients of the equation for $b$ creates a very large perturbation of line $g$. It is much better to select the lines $a$, $d$, $e$, and $f$ as type 1 lines. In view of this, the following approach yields an algorithm for polygon intersection that is likely to yield practically satisfactory results for polygon intersection: Consider one polygon exact as written, i.e., use the natural model for it. Now perturb the edge positions of the other polygon by trying to satisfy first those near-incidences on an edge that are farthest apart. If this distance is small such that the resulting vertex position would be perturbed by more than a specified maximum distance, then drop one of the constraints. Again, one can implement this algorithm with bounded precision arithmetic.

## 6. Robustly Computing the Intersection of Two Polyhedra

The intersection of two polyhedra can be obtained by a sequence of polygon intersections. Two types of difficulties arise in this approach. In certain situations we are dealing with more than two polygons simultaneously. The other difficulty is that line segments belonging to different polygons may arise from the same face and thus cannot be adjusted independently.

Consider the intersection of an arbitrary polyhedron with a convex polyhedron. There is a surprising degree of flexibility in the definition of correctness. From a mathematical point of view, the intersection of a convex polyhedron $P_1$ with an arbitrary polyhedron $P_2$ is equivalent to intersecting $P_2$ with the set of halfspaces defining the convex polyhedron. However, with approximate representations, intersecting $P_1$ and $P_2$ differs from intersecting $P_2$ with each of the halfspaces defining $P_1$. In the first case, given representations $R_1$ and $R_2$, $R_3$ is a correct result if there exist corresponding models $M_i$ such that $M_3 = M_1 \cap M_2$. In the second case, the definition of correctness for a halfspace representation $R_H$ and a polyhedron representation $R_1$ is that there exist corresponding models $M_1$ and $M_H$ such that we obtain an output representation $R_2$ with model $M_2$ such that $M_2 = M_1 \cap M_H$. The intersection of $R_1$ and $R_2$ is then obtained by successively intersecting with halfspaces. A representation $R_n$ that is a correct intersection by the second definition need not be correct for the first definition since intersection is not associative. Whatever definition is adopted, it must yield valid objects that agree with the ordinary set theoretic intersection for objects none of whose features coincide or nearly coincide. Moreover, it must be implementable in a provably correct manner.

The usefulness of the second definition is that it can be implemented in a provably correct fashion. When intersecting with a halfspace, we must determine for each vertex of the polyhedron on which side of the plane that bounds the halfspace it lies. Numerical computation suffices for certain vertices. If the polyhedron is trihedral, we can arbitrarily place the other vertices on one side or the other, except that if several vertices of the same face are near the plane then we must place them in a consistent manner. For example, we cannot claim that three noncollinear vertices of a face are on the plane and a fourth vertex of the same face is off the plane. However, since the output polyhedron need not be trihedral, this approach does not lead to an algorithm for intersecting a trihedral and a convex polyhedron.

The halfspace intersection approach requires one of the polyhedra to be convex. A better algorithm that can be extended to the intersection of arbitrary polyhedra $P_1$ and $P_2$ is as follows: Intersect the plane of each face of $P_1$ with solid $P_2$ to obtain a set of cross sectional graphs. Each cross sectional graph is clipped by the face of $P_1$ associated with the plane that gave rise to the cross section. Similarly intersect the plane of each face of $P_2$ with solid $P_1$ and clip the cross sectional graphs with the appropriate face of $P_2$. The representation of $P_1 \cap P_2$ is then constructed from these cross sections.

Constructing the cross section is analogous to intersecting polyhedra with a half space. Clipping the cross sections, however, presents added difficulties. First, if the plane cuts $P_2$ so that the cross section contains a face, an edge, or a vertex of $P_2$, then the cross sectional graph will have a structure that represents two cross sections of $P_2$; i. e., the cross section on each side of the plane. Thus the cross section is equivalent to superimposing two polygons and clipping gives rise to a third. Figure 8 shows a polyhedron and one of its cross sections. Clipping with the polygon shown again introduces a complexity equivalent to Pascal's theorem. In the case where one of the polyhedra is convex, the solid on one side of the plane was discarded, as described above. This reduced robust clipping to intersecting two polygons. When neither polyhedron is convex we can simplify clipping by discarding edges of the cross sectional graph that arise solely because of the structure of the solid on the side of the plane determined by the positive face normal. This reduces the cross sectional graph to a collection of polygons intersecting only at vertices and hence reduces the clipping problem to the polygon intersection problem which can be done robustly.

Two problems arise. The first has to do with constraints on the edges of the polygons involved. For example, in the cross sectional graph, it may be the case that several edges arise from the intersection of the cross sectional plane with the same face of the solid. In this case the resulting edges must be on the same infinite line. These additional constraints may not permit robust clipping. Note, however, that the problem can be resolved, as shown in Figure 9, by partitioning the face.

The second problem is one of global consistency.

Although each cross sectional graph can be clipped robustly, we must make sure that they are clipped consistently, as explained next.

## 7. Clipping Different Cross Sections Consistently

Given two faces $F_1$ and $F_2$ we must insure that the cross sectional graphs generated by the planes of $F_1$ and $F_2$ are clipped in a consistent manner. Since an edge $a$ of $F_1$ and an edge $b$ of $F_2$ may be generated by the same face $F_3$, they cannot be reoriented independently in the respective planes (Figure 10). In addition, a face of the other solid may intersect the planes containing $F_1$ and $F_2$ simultaneously, and thus its intersection lines may also not be moved independently. Both types of constraints must be accounted for. They become especially delicate when an edge $e'$ of the polyhedron $P_2$ intersects a face of polyhedron $P_1$ near an edge $e$ of the face. Here, the edge $e'$ intersects the face plane in a vertex of the cross section graph, and we must specify where this vertex lies with respect to the face boundary $e$. Further complications arise in the vicinity of a vertex of $e$, and a detailed case analysis is required. See also [6].

## 8. Discussion

We have presented several paradigms for correctly implementing a variety of geometric computations. The reasoning paradigm considers the numerical information to be approximate to real data, and seeks to derive information from the symbolic data describing adjacencies. As we showed, the reasoning component varies considerably with the geometric structure of the input: Intersecting two polygons is easy, but intersecting simultaneously several polygons requires proving theorems from projective geometry. So far, we have been unable to prove correctness of a polyhedral intersection implementation, but we feel that this approach will succeed. We have implemented a polyhedral intersection algorithm based on these ideas and have tested it in a variety of cases. For example, a unit cube was intersected with randomly rotated copy of itself. The resulting polyhedron was in turn intersected with a randomly rotated copy of itself, and so on. After twelve iterations, the polyhedron shown in Figure 11 was obtained; see also [5,6]. When intersecting polyhedra with a rotated copy, angles as small as 1/10,000 of a degree have been used. As the angle of rotation is diminished, the algorithm starts to consider near-coincident features to be coincident. Below a certain threshold, the algorithm declares the two copies to be identical. Experimental evidence given in [6] suggests that the algorithm gives a topologically correct result for all rotation angles except those in a very small range. Depending on the particular experiment, this range has been as large as $10^{-5}$ degrees and as small as $10^{-10}$ degrees.

Even though the reasoning paradigm is logically satisfactory, it may not have very good numerical behavior and may lead to large perturbations. The placement strategy of Section 5 strikes a compromise in that some numerical data is taken as accurate while other data is perturbed. This approach seems to produce smaller perturbations than the reasoning paradigm. Nevertheless, in practice this has not been a problem, and the paradigm has led to a polyhedral intersection algorithm that is substantially more robust than the algorithms previously reported in the literature.

The exact-as-written paradigm of Section 5 is very satisfactory for simple objects such as line segments. It has been used for provably correct polyhedron intersection [10], but has a number of draw-backs. Briefly, it is not possible to rotate or translate such a polyhedron without reconstructing it from the rotated or translated primitives, due to the presence of very small features. Moreover, it seems that this paradigm cannot be extended to nonlinear geometric objects: The intersection point of linear structures with rational coefficients has rational coordinates, but the same is not true for nonlinear structures. Finally, the proliferation of small features is not desirable in many applications.

## 9. References


[1] H. Durrant-Whyte (1986) "Concerning Uncertain Geometry in Robotics," *International Workshop on Geometric Reasoning*, Oxford, England, July 1986.

[2] H. Edelsbrunner, J. O'Rourke, R. Seidel (1986) "Constructing Arrangements of Lines and Hyperplanes with Applications," *SIAM J. Comput. 15*, 341-363.

[3] G. Golub and C. van Loan (1983) "Matrix Computation," Johns Hopkins Press, Baltimore.

[4] D. Greene, F. Yao (1986) "Finite-Resolution Computational Geometry," *Proc. 27th IEEE Symp. on Found. Comp. Sci.*, Toronto, 143-152.

[5] C. Hoffmann, J. Hopcroft, M. Karasick (1987) "Robust Set Operations on Polyhedral Solids," Tech. Rept. 87-875, Comp. Science, Cornell University

[6] M. Karasick (1988) "On the Representation and Manipulation of Rigid Solids," PhD thesis, McGill University, Montreal.

[7] J. Lagarias (1985) "The computational complexity of simultaneous Diophantine approximation problems," *SIAM J. Comp. 14*, 196-209

[8] V. Milenkovic (1986) "Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic," *International Workshop on Geometric Reasoning*, Oxford, England, July 1986.

[9] T. Ottmann, G. Thiemt, C. Ullrich (1987) "Numerical Stability of Geometric Algorithms," *Proc. 3rd Symp. Comp. Geometry*, Waterloo, 119-125.

[10] K. Sugihara (1987) "An approach to error-free solid modeling," Notes, Instit. for Math. and Applic., Univ. of Minnesota

[11] K. Sugihara (1987) "On finite precision representations of geometric objects," Res. Memo RMI 87-06, Dept. of Math. Engr., Univ. of Tokyo

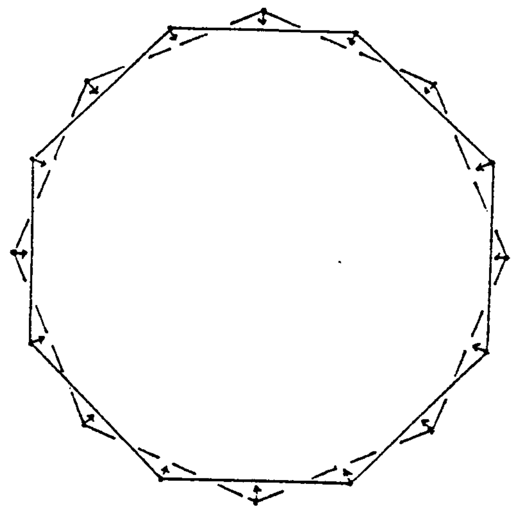

Figure 1

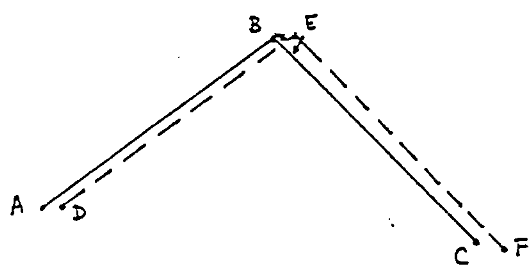


Figure 2

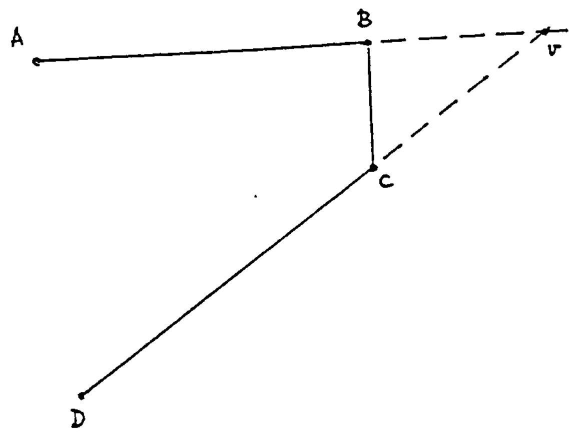


Figure 3

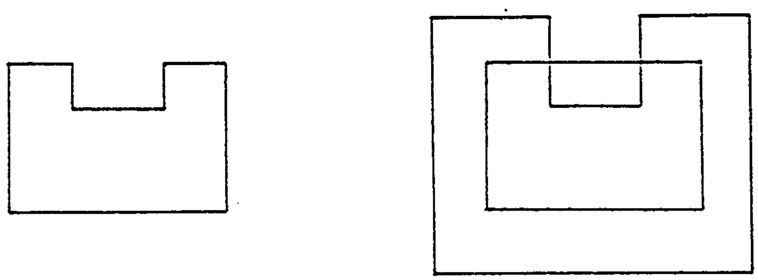

Figure 4

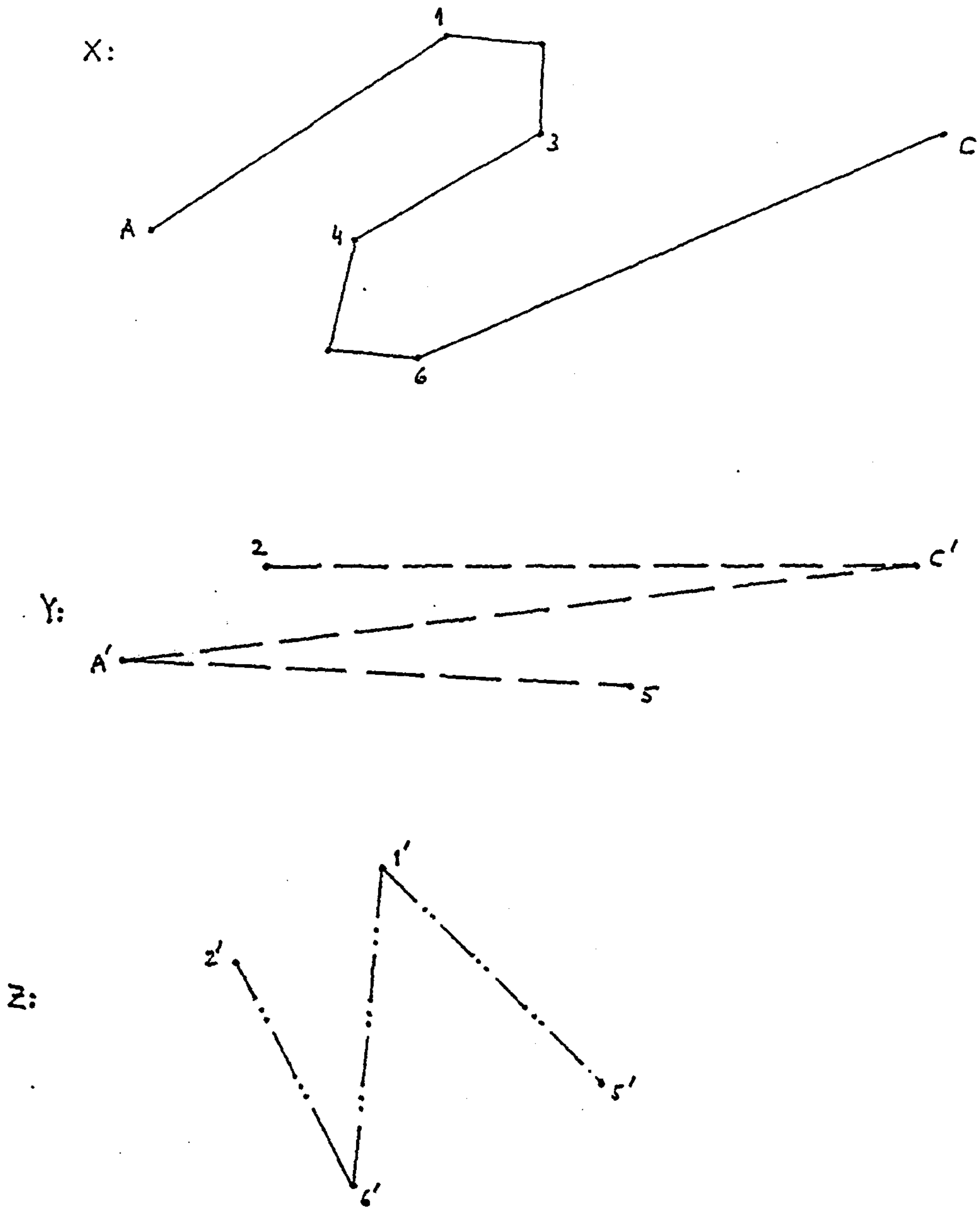


Figure 5

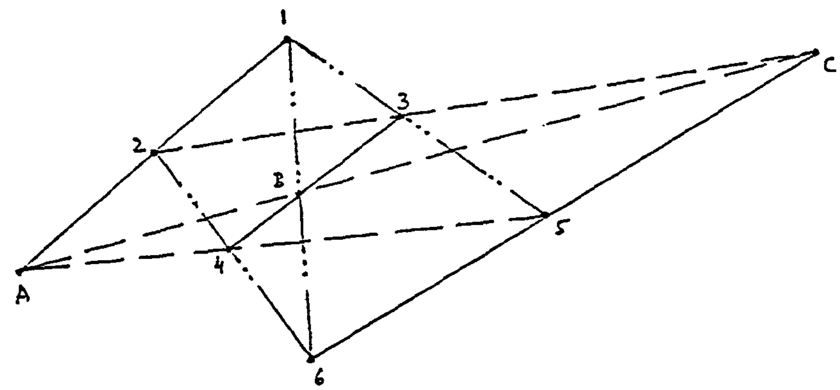


Figure 6

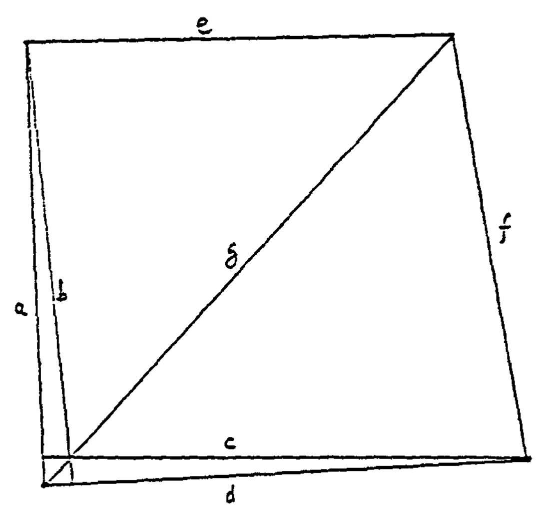


Figure 7

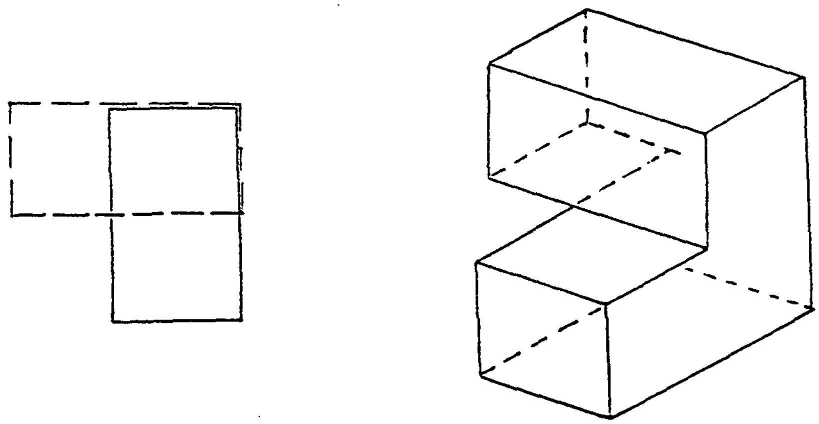

Figure 8

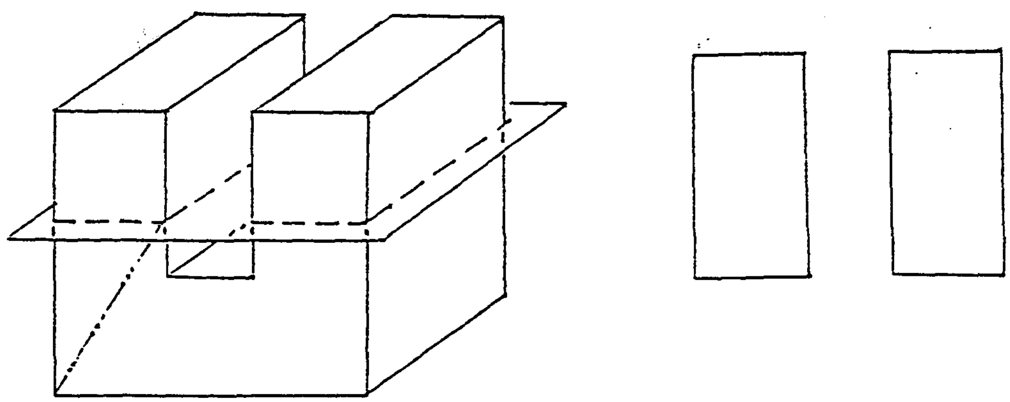

Figure 9

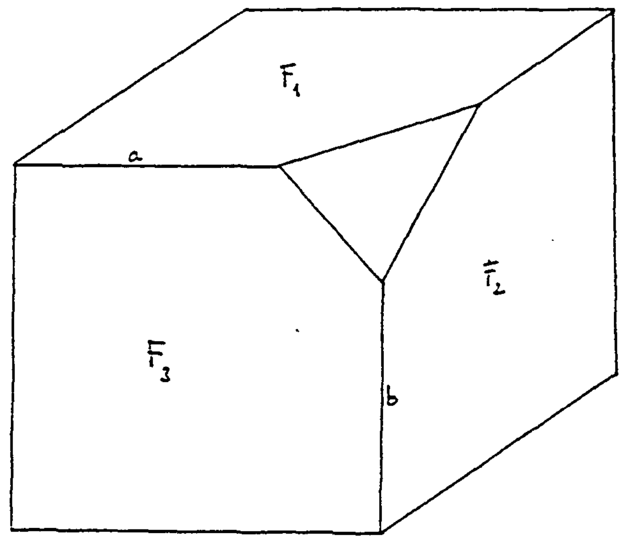


Figure 10

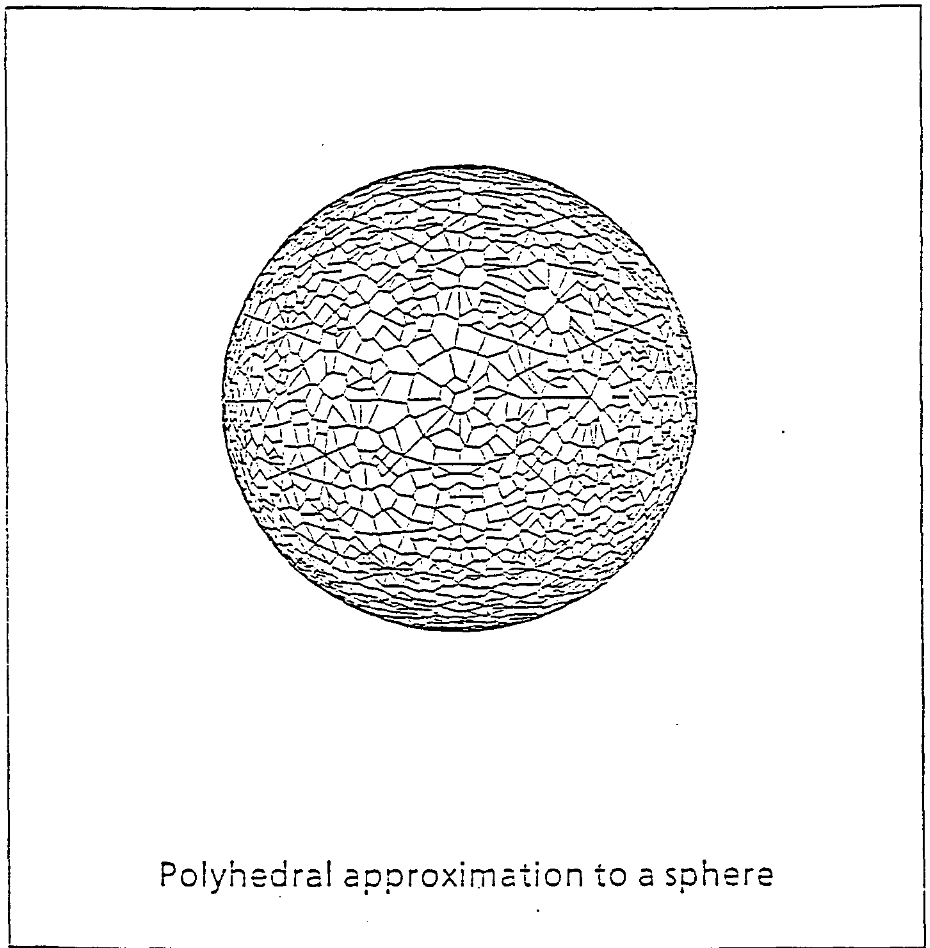


Figure 11