1988

The Problem of Accuracy and Robustness in Geometric Computation

Christoph M. Hoffmann
Purdue University, cmh@cs.purdue.edu

Report Number:
88-771

THE PROBLEM OF ACCURACY AND ROBUSTNESS IN GEOMETRIC COMPUTATION

Christoph M. Hoffmann

CSD-TR-771
April 1988

The Problem of Accuracy and Robustness in Geometric Computation

Christoph M. Hoffmann*

Abstract

Except in very simple cases, input and output of geometric algorithms are data structures containing both numerical and symbolic data. While it is fair to assume that the symbolic data is exact, the numerical data usually is not. In consequence, the geometric algorithm must account for the possibility of internal data inconsistency, and foundational techniques must be developed that are capable of drawing meaningful conclusions. We will sketch the dimensions of this problem and outline a number of approaches that might succeed in dealing with it.

1 Introduction

Despite great advances in geometric and solid modeling, practical implementations of the various geometric operations remain error-prone, and the goal of implementing correct and robust systems for carrying out geometric computation remains elusive. This fact seems to be due to an underlying difficulty that sets geometric computation apart from other application areas in computer science and engineering. Recently, a number of publications have begun to address this question, but there is no general agreement as to the true source of the problem, and about what strategies might succeed in solving it. In particular, the problem is variously characterized as a matter of achieving sufficient numerical precision, as a fundamental difficulty

*Computer Science Department, Purdue University, West Lafayette, Ind. 47907. Supported in part by NSF Grant CCR 86-19817, ONR Contract N00014-86-K-0465, and a grant from the ATT Foundation.

in dealing with interacting numeric and symbolic data, or as a problem of avoiding degenerate positions. Here, *degenerate position* could refer to positional incidence, or to tangential, as opposed to transversal, intersection.

In fact, these issues are interrelated, and are rooted in the problem that objects conceptually belonging to a continuous domain are analyzed by algorithms doing discrete computation, treating a very large discrete domain, e.g., the set of all representable floating-point numbers, as if it were a continuous domain. There are many situations in which this approach leads to acceptable results, but it appears that except for very simple geometric objects this simplification does not work very well, and that it expresses itself in occasional failure of the implemented algorithm operating on correct inputs. The purpose of this paper is to examine these problems, to survey the various approaches proposed, and to assess their potential for devising complete and efficient solutions. We restrict our analysis to objects with linear elements, since substantial problems already arise in this case.

Linear geometric objects, in three dimensions, usually consist of points, edges, and polygonal faces, that are in specific spatial relationship to each other. The specification of such objects consists then of two parts: numerical information, recording, e.g., vertex coordinates or plane equations, and symbolic data specifying face and edge boundaries, adjacencies, and incidences.

Usually, the arithmetic data describing a geometric object is given only approximately, using, e.g., a floating-point representation. In consequence, the imprecise results may lead to contradictory information about the object that is input. For instance, the object description may require that four adjacent faces meet in a common vertex, yet the numerical plane coefficients for the faces may specify four planes that intersect in four closely-spaced points, rather than one single one.

In many geometric operations, the result of numerical computation must be used to infer symbolic fact, e.g., that a vertex of A is incident to a face of B. Here, using limited precision arithmetic has the consequence that the outcome of the computation may crucially depend on the detailed sequence of calculations. Since there is uncertainty associated with approximate arithmetic computation, some decisions are made with incomplete information. Different decision so made must be logically consistent. Moreover, there are certain symbolic facts that can be determined by different sequences of arithmetic calculation which could result in contradictory answers. For

example, determining whether a vertex u of A coincides with a vertex v of B can be done as follows:

1. Compute the distance of u from each of the planes intersecting in v.
2. If the distance from all planes is smaller than a fixed tolerance, then declare that u and v are coincident.

In this computation, interchanging the role of u and v could result in contradictory conclusions, e.g., we might determine that u is incident to v but not vice versa.

Given these possibilities, it must be asked what it means that a geometric algorithm is correct, and that it delivers an acceptable result for all legitimate inputs. Indeed, the implementations of virtually all geometric algorithms will fail for certain correct inputs. In this paper, we examine the problem, explore a number of approaches that have been suggested, and comment on their potential.

2 Floating Point Computation

Using a very simple example, Dobkin and Silver [2] have illustrated the difficulty of accurately carrying out geometric computations. They consider a pentagon A in the plane. Drawing the five diagonals of A, their intersections define an inscribed pentagon B. Let us call the operation of passing from A to B as *going in*, and write symbolically $B = in(A)$. Similarly, we can extend the five sides of A to their outer intersections, thus obtaining a circumscribing pentagon C. We call this operation *going out*, and write $C = out(A)$. Clearly $A = out(in(A))$ and $A = in(out(A))$. Taking a pentagon A, they iterate the *going in* operation m times, obtaining $B = in^m(A)$, and then compute $C = out^m(B)$. Then they compare the coordinates of the vertices of A and of C. Ideally, they should be equal. In practice, they can differ by a large error even for values of m as small as 2 or 3.

As illustration, let us consider a pentagon with vertices at $(0,0)$, $(1,0)$, $(0,1)$, $(1+p,1)$, and $(1,1+p)$, for several values of m and of p. See also Figure 1. Table 1 shows the results, with all computations performed in single precision IEEE standard floating point arithmetic. The table demonstrates dramatically that the numerical output from very simple geometric

p	$out^2(in^2())$	$out^3(in^3())$	$in^2(out^2())$	$in^3(out^3())$
0.1	$3 \cdot 10^{-5}$	$3 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	$4 \cdot 10^{-5}$
0.01	$2 \cdot 10^{-3}$	$9 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	$7 \cdot 10^{-2}$
0.001	$2 \cdot 10^{-1}$	2	15	45
0.0001	$5 \cdot 10^{-1}$	∞	$3 \cdot 10^{7}$	∞
0.00001	∞	∞	$6 \cdot 10^{8}$	∞

Table 1: Absolute Error for Iterating *going in* and *going out* Operations

operations can be quite inaccurate. [2] suggests extending the precision of the calculation as needed, until the calculation achieves an acceptable output precision for the specific problem. Let us consider the consequences of this suggestion.

The two operations iterated on the pentagon involve finding the equation of the line containing two given points u and v, and computing the coordinates of the intersection of two lines. Of these computations, finding the intersection of two lines,

$$\begin{aligned} a_1 x + b_1 y + c_1 &= 0 \\ a_2 x + b_2 y + c_2 &= 0, \end{aligned}$$

is the more delicate one, as it involves inverting the matrix

$$A = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix}$$

The accuracy with which the matrix can be inverted can be estimated by the *condition number*

$$\kappa = \| A \|_\infty \| A^{-1} \|_\infty$$

See, e.g., [5]. Roughly speaking, the larger the condition number, the more the solution will be sensitive to random perturbations of the equation coefficients. Such perturbations will occur in floating point arithmetic due to rounding and cancellation errors. Simplifying somewhat, we estimate that a relative perturbation of the line coefficients by ϵ leads to a relative perturbation of the point coordinates proportional to $\epsilon\kappa$.

The condition number of the matrix A can be expressed in terms of the angle between the two lines. Assume that one of the lines has a 45° slope. With β the angle between two lines, we can show that the condition number is

$$\kappa(\beta) = \frac{1}{\sin(\beta/2)}$$

Under extremely favorable circumstances, we expect a perturbation of the input coefficients of order 2^{-t}, where t is the machine precision. That is, the computed result will differ from the true result only by roundoff in the last representable digit. For small angles β we have $\sin(\beta/2) \approx \beta/2$. Hence, for an intersection angle of $1/2^m$ we expect to lose approximately $m+1$ binary digits.

A specific difficulty with geometric computation is the fact that we want to subject a geometric object to several operations in sequence, that is, that these computations are iterated. Hence, the precision loss is additive, e.g., when iterating computations such as the *going in* and *going out* operations.

We demonstrate the loss of precision due to small perturbation. Assume that the line intersection (u, v) is computed as follows:

$$\begin{aligned} D &= a_1 b_2 - a_2 b_1 \\ u &= (c_2 b_1 - c_1 b_2)/D \\ v &= (a_2 c_1 - a_1 c_2)/D \end{aligned}$$

We use the pair of lines:

$$\begin{aligned} -x + y &= 0 \\ -(1+q)x + (1-q)y + 2q &= 0 \end{aligned}$$

With $q = 1/2^m$ and $m > 5$, these lines intersect exactly in the point $(1,1)$, at an angle q of less than one degree. Moreover, as long as m does not exceed the mantissa length, the coefficients are exact in floating-point representation. With these precise numbers, the point $(1,1)$ is determined without error by floating point arithmetic. We then perturb the coefficients by $p = 1/2^t$, choosing t to machine precision. Specifically, we solve the following perturbed system, expected to lead to the largest deviations with a coefficient error no greater than machine precision:

$$\begin{aligned} -(1+p)x + (1-p)y &= 0 \\ -(1+q-p)x + (1-q+p)y + 2q &= 0 \end{aligned}$$

With $q = 1/2^{18}$, the two lines intersect at approximately one arc second, resulting in a condition number of 2^{19}. So, we expect to lose about 19 significant binary digits when perturbing the system by machine precision. Indeed, while the computation for the unperturbed system yields $(1.0, 1.0)$ exactly, the perturbed system, with $t = 23$, yields the point $(1.07, 1.07)$, in good agreement with predictions.

We now return to the pentagon problem. Suppose we could guarantee an angle of no less than one arc second, between any pair of lines encountered during the iteration. Then computing *going in* three times followed by three *going out* operations would require doing the calculation at more than triple precision, to guarantee a meaningful result. Quadruple precision calculation would yield, in the worst case, only 14 significant binary digits, i.e., a little more than four decimals. Note that the internal precision needed depends on

1. an estimate of the minimum occurring angle, and
2. fixing in advance the sequence of operations to be done.

Both factors may be unknown for the geometric problem at hand.

Dobkin and Silver suggest computing an accuracy estimate for each geometric calculation. Rather than using the condition number, however, the inputs to the computation are systematically perturbed and the effect on the output is measured. That is, the inputs are perturbed and the computation repeated. [2] states that 3 or 4 different perturbations suffice to obtain an accurate estimate of the actual output precision. Such an analysis allows also, in principle, an estimate of the internal precision needed to solve a problem *instance* with a desired accuracy.

3 Purely Symbolic Representations

We consider whether a symbolic representation can avoid the numeric problems completely, while not introducing new and difficult problems. To this purpose, we investigate geometric objects consisting of *lines*, given as $[a, b, c]$, and *points*, given as (u, v, w), where a, b, c, u, v, and w are symbols drawn from a fixed universe Σ^*. Here, the triple $[a, b, c]$ symbolizes the line equation $ax + by + cz = 0$, and the triple (u, v, w) projective point coordinates. Recall

that the projective point (u, v, w) corresponds to the affine point $(u/w, v/w)$ whenever $w \neq 0$.

Specifying that the point $P = (u, v, w)$ is *incident* to the line $L = [a, b, c]$ shall mean that the equation $au + bv + cw = 0$ can be satisfied, and we write this fact as $L(P)$. We specify an arrangement of points and lines by the following rules:

(D1) All lines and points must be declared in advance, as triples of symbols. No two lines and no two points so declared are equal.

(D2) If a point P is incident to a line L, then this fact is explicitly stated as $L(P)$. If two lines L_1 and L_2 intersect in the declared point P, then this fact is expressed explicitly by the two incidence statements $L_1(P)$ and $L_2(P)$.

(D3) No other incidences exist among declared points and lines except those explicitly stated.

These rules have been closely modeled after common conventions in boundary representations of polyhedra, where one requires that vertices, edges, and faces are all distinct, and that they do not intersect except in explicitly stated adjacencies.

Given a symbolic object specification in the above methodology, we investigate whether it can be realized as a point/line configuration in real two-dimensional projective space $\mathbf{P}^2$. That is, we ask whether there exists an assignment of real numbers to the symbols such that

1. The equations entailed by (D2) above are satisfied, and
2. all points and lines are distinct and satisfy (D3).

Note that this question differs from the traditional, combinatorial question whether there exists an abstract projective geometry satisfying the required conditions, since we insist on an embedding into the *specific* geometry $\mathbf{P}^2$.

It is not hard to find an example of a description that cannot be realized in $\mathbf{P}^2$: Consider the following configuration consisting of nine distinct points,

$$P_1 = (u_1, v_1, w_1), \ldots, P_9 = (u_9, v_9, w_9)$$

and of nine distinct lines

$$L_1 = [a_1, b_1, c_1], \ldots L_9 = [a_9, b_9, c_9]$$

The required incidences are as follows:

$$\begin{array}{llllll} L_1(P_1), & L_1(P_3), & L_1(P_5), & L_2(P_2), & L_2(P_4), & L_2(P_6), \\ L_3(P_1), & L_3(P_2), & L_3(P_7), & L_4(P_2), & L_4(P_3), & L_4(P_9), \\ L_5(P_3), & L_5(P_4), & L_5(P_8), & L_6(P_4), & L_6(P_5), & L_6(P_7), \\ L_7(P_5), & L_7(P_6), & L_7(P_9), & L_8(P_6), & L_8(P_1), & L_8(P_8), \\ L_9(P_7), & L_9(P_8), & L_9(P_9) & & & \end{array}$$

This configuration exists in $\mathbf{P}^2$ and is shown in Figure 2. However, if the last incidence constraint, $L_9(P_9)$, is removed, then there is no such configuration in $\mathbf{P}^2$, since it contradicts Pascal's Theorem.

This example demonstrates that a purely symbolic representation raises logical existence problems. Such problems would require symbolic reasoning, and one of the questions it must answer is the logical power such a reasoning algorithm would have to have. This logical power depends on the domain of geometric objects. As we shall see below, the reasoning algorithm could be quite simple for polygonal intersection. For incidence configurations in projective geometry, it requires considerably more machinery, and partial results exist in the area of geometric theorem proving. See, e.g., [1]. In the case of polyhedra, it raises a number of open problems; see also Section 4 below.

Note that this example also has implications for algorithms doing numerical computation to determine incidence: Assume that we are given the nine lines $L_1, \ldots L_9$ by their equations and we want to compute how they intersect. Then the incidences listed explicitly above would have to be inferred for the line intersection points, based on numerical computation. In that case, we would have to decide whether three lines, say L_1, L_3 and L_8, intersect in a common point. Unless we compute with exact numbers, however, the three lines are likely to intersect in three closely spaced, distinct points. A decision of whether they should be coincident, therefore, will depend on other such decisions, for other sets of lines, and not solely on the result of the numerical computation.

4 Representation and Model

If we accept the possibility of imprecise numerical data, then we need to explain first what is described by such a representation. Only after giving to such a representation a precise geometric meaning, can we decide whether or not a geometric algorithm has been correctly implemented. Thus, we introduce the notions of *representation* and *model*.

A *representation* is a description of a geometric object, possibly using imprecise arithmetic data. Such a representation has a *model*, if there exists an object in Euclidian space satisfying the symbolic part of the description precisely, and whose numeric data is exact, although it might require infinite precision numbers. As we have seen above, the question whether there exists a model for a given representation is delicate, and must not be dismissed lightly.

As example, consider the representation of a cube whose symbolic data specifies only the topology of vertices, edges and faces, but makes no mention of the fact, e.g., that the faces are square and that opposite faces are parallel. Then any six-sided trihedral polyhedron with four-sided faces will be a model, irrespective of the approximate vertex coordinates given in the representation. It is clear that this definition of model is too broad to be useful, so we attempt to capture the intent of the representation better.

Clearly, the numerical data in the representation is intended to be close to the exact data. Therefore, it makes sense to compare the exact data of the model with the approximate data of the representation:

A model M of a given representation R is *ϵ-close*, if the largest deviation of the numeric data of the representation from the exact model data is no greater than ϵ. This is an absolute error notion which suffices for the purposes of this paper, but it could clearly be replaced by a relative error definition.

With these concepts, we can now clarify when a k-ary geometric operation *op* is correctly implemented: The implementation of *op* is *correct*, if for every input representation R_i there exists a model M_i such that the following is true:

1. The algorithm constructs an output representation R without failing.
2. There is a model M of R such that $M = op(M_1, ..., M_k)$.

The definition can be further specialized to capture the precision of the algorithm as follows: Given that each model M_i is ϵ-close to its representation, the model M is $\delta(\epsilon)$-close to R.

Clearly, one desires assurance of the existence of a model M for all legitimate input data, and a function δ such that $\delta(\epsilon)$ is not excessively large compared to ϵ.

In the case of polyhedra, it is common to assume that the representation describes a model that is ϵ-close to a given region, and the ϵ is sometimes explicitly specified, e.g., [10]. This is often expressed by saying that there is a "fuzz region" enveloping the surface, and that the intended exact polyhedron lies within this fuzz region. From a mathematical point of view, this appealing intuitive concept is defect, and there are no theoretical results that guarantee the existence of such a model, unless further restrictions are placed on the topology of the polyhedron. For instance, Hopcroft and Kahn (private communication) prove the following theorem: Given a winged-edge boundary representation of a trihedral polyhedron in which the intersection of every pair of adjacent faces is connected, there exists a model satisfying the topology exactly.

5 Limited Precision Rational Arithmetic

In [11], Sugihara has proposed to use exact rational arithmetic of bounded precision. Sugihara approaches Boolean operations on polyhedral solids as follows:

1. In the representation of a polyhedron, only plane equations are given numerically, in the form $ax + by + cz + d = 0$, where a, b, c, and d are integers. All other information is symbolic: Vertices are given as intersection of three or more planes, edges as intersection of two planes, and so on.

2. Any polyhedron is built from a sequence of operations on primitives, and these primitive polyhedra must be trihedral and must satisfy a minimum feature condition.

To limit the precision of the rational numbers occurring throughout the geometric computations, Sugihara requires that the magnitude of a, b, and

c is less than a maximum L, and that the magnitude of d is less than L^2. The reason for giving a different bound on the magnitude of d is illustrated in two dimensions in Figure 3. He observes that the grid of representable planes is more uniform with $|d| < L^2$ than with $|d| < L$.

Primitive polyhedra must be trihedral, i.e., exactly three faces are incident at every vertex. Moreover, no edge is shorter than, and no vertex closer to any edge or face, than a fixed minimum.

When so limiting the set of representable planes, it is clear that there is an absolute minimum distance between any point u that is the intersection of three planes and a fourth plane P, whenever u is not on P. This minimum distance gives rise to an estimate of the internal precision needed to test correctly incidence or nonincidence of geometric structures. If the input precision is L decimal digits, intermediate integers with up to $5L+2$ decimal digits will be needed, since a vertex $u = P_1 \cap P_2 \cap P_3$ is on a plane P_4 precisely when the determinant

$$\begin{vmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{vmatrix}$$

is zero.

Sugihara's approach deals successfully with the problem of possible rapid growth in the number of digits needed to represent numeric data in a sequence of geometric operations. When using it to implement a polyhedral modeler, however, it has the drawback that rotating or translating a complex polyhedron is not straightforward. To appreciate this problem, consider Figures 4 – 6. Figure 4 shows the union of 150 triangles randomly generated with one vertex on a circle of radius 10^{-3}, the other two on the unit circle. One observes that the boundary of the resulting object contains many small features, such as the narrow crack shown in magnification in Figure 6. When translating or rotating the object, there is no guarantee that in the new position there exist representable grid lines that can bound such a feature. In fact, upon rotation or translation, each line may have to be 'rounded' so that its coefficients conform to the limits in magnitude. Possibly, then, the feature would be altered to look as shown in Figure 7, so that a simple polygon might be changed to one that is not simple.

In view of this problem, Sugihara suggests the following: When translating or rotating a complex polyhedron P, separately translate or rotate

the primitives from which P has been built, and then reconstruct P from the resulting primitive objects. Since primitives are trihedral and satisfy a minimum feature separation condition, each translated or rotated primitive remains topologically valid, although it may experience small changes in shape. For example, when rotating a cube, the faces need not remain square or parallel, although they can be expected to remain nearly so. It is clear that the possible slight deformation of the primitives also results in a change of the topology of P. That is, the translated or rotated version of P may well differ from P in the number of vertices, faces and edges.

Both, the need to reconstruct a polyhedron P and the fact that a move of P alters its topology are limitations of this approach. Moreover, it is not clear how the approach can generalize to geometric objects with nonlinear elements. Two difficulties intrude: While the intersection of planes with integer coefficients is a rational point, the intersection of quadric surfaces with integer coefficients need not be rational. Already in two dimensions, the intersection of $x^2+y^2-1=0$ and $x-y=0$ is irrational. Moreover, while again a minimum distance must exist between any two representable distinct points, estimates of the intermediate precision needed to separate them become very unfavorable. Sugihara estimates the needed internal precision for objects with quadric surfaces at 80 times the input precision.

6 Reasoning Paradigm

Assume we implement a geometric algorithm using floating point arithmetic. We know that we must deal with imprecise numeric data, and, for some numeric computations, it will be uncertain as to what conclusions may be drawn from the results. By carefully analyzing the condition number of each calculation, we can establish the following scenario:

> A numerical computation C is carried out. Depending on whether the outcome is positive, zero, or negative, a logical decision must be made, on which subsequent processing depends. As long as the magnitude of the result r exceeds a certain threshold $\tau(C)$, we can make the decision with certainty. If the magnitude of r is smaller than $\tau(C)$, the decision based on r alone is uncertain.

When the decision is uncertain, we could make it arbitrarily, e.g., we could require that a result r of magnitude $|r| < \tau$ is understood to mean $r = 0$. But the fact that such a decision could have consequences for other, later decisions obliges us to make each decision in a logically consistent manner.

As example, consider the problem of deciding if a vertex v of a polyhedron B is incident to a face, edge or vertex of a polyhedron A. Suppose we have already determined that v lies in the plane containing the face f of A, and that it is so close to an edge e of A that v is probably on e. If we now decide that v is incident to e, then we must also decide that v is incident to the face f' of B that is adjacent to f by the edge e. Although this example may seem trivial, it is one of the specific numerical problems that cause failure of many polyhedral modelers for certain inputs. One of the major problems, then, is to determine how difficult the reasoning must be by which the algorithm establishes that the decision to be made next is consistent with all prior decisions. This problem has been explored in [8].

In order to approach this reasoning problem, we need a strategy for establishing that the algorithm is correct based on the decision-making process that interprets the meaning of inconclusive numerical computations. Using the operation of intersection as an example, we need to show that the implementation has the following properties:

1. The algorithm produces a valid output representation R for all valid input representations R_1 and R_2.

2. There exist models M_1, M_2, and M of the representations R_1, R_2, and R, respectively, such that $M = M_1 \cap M_2$.

3. If M_1 and M_2 are ϵ-close to R_1 and R_2, then M is $\delta(\epsilon)$-close to R.

An important step is to establish the existence of the models M, M_1, and M_2 satisfying $M = M_1 \cap M_2$. The utility of the resulting implementation, however, will also depend on the quantification of the tolerances ϵ and $\delta(\epsilon)$. In [8], we have shown that the intersection of two polygons can be computed in this sense with essentially no reasoning. That is, it was shown that no matter how incidence is decided, except, possibly the last such decision, that models M, M_1 and M_2 exist satisfying $M = M_1 \cap M_2$. Technically, this involves showing that the various edges can be suitably repositioned so that all required incidences exist in the models.

In Section 3 we saw that incidence requirements can lead to difficult reasoning problems. Polygonal intersection is free of these difficulties as long as we do not require satisfaction of additional positional properties, such as the collinearity of nonconsecutive edges. These constraints can be introduced artificially, by considering the *simultaneous* intersection of three or more polygons.

For polyhedral intersection considerable difficulties arise, since it is no longer evident how to reposition a face consistently, so as to satisfy all incidence decisions. As example, consider the polyhedron A shown in Figure 8. Assume we need to adjust the plane containing the face f to accommodate some incidence decisions we made when analyzing the position of A with respect to some other polyhedron B. Since we must preserve the planarity of f, at most two vertices can remain in the original position. At the other vertices, therefore, the shape and, possibly, the position of adjacent faces must also be changed, so as to preserve the topological structure. In consequence, the operation of repositioning a face requires a global alteration of the polyhedron. Whether such an alteration can be carried out, i.e., whether there exists a model of the altered polyhedron representation, is not clear. For this reason, we can prove that a trihedral polyhedron can be correctly intersected with a half space, but we cannot conclude from this that a trihedral polyhedron can be intersected with a convex polyhedron, since the result of the first intersection need not be trihedral.

The stringent requirement that the topological data agree between representation and model also affects what bounds can be established on the closeness δ of the output model M, as function of the closeness ϵ of the input models M_1 and M_2. The reason for this is foremost a technical one: As stated above, when proving the correctness of an implementation in this framework, we have to show that the elements of the input models can be consistently repositioned so as to satisfy the incidence constraints introduced during the course of the computation. In all likelihood, this repositioning is sequential, for proof purposes. But the repositioning sequence affects the final position of the vertices, edges and faces. As example, consider the configuration shown in Figure 9. Here, positioning vertices in the order 1, 3, 4, 5, 2, 7, 6 is much more favorable than positioning them in the order 1, 2, 7, 6, 3, 4, 5, since a small position perturbation of the vertices 2, 6, and 7 leads to a large perturbation of the vertices 3 and 5. Thus, not only would we like to show that a consistent sequence of repositioning operations exists, for all inputs and all incidence decisions on it, but also that the specific sequence

chosen leads to small positional perturbations.

7 Altering the Symbolic Data

So far, we have required that the symbolic data of the representation is literally satisfied by the model. Thus, the symbolic data is considered more trustworthy than the numeric data, when deducing the intended meaning of a given representation. The rationale for giving priority to the symbolic data is that it can be represented easily without error. Assuming that there exists a reliable method for defining geometric objects, the symbolic data in the representation ought to be correct as given.

However, objects are often constructed from other objects by geometric operations, and so there is a chance that the implementation has introduced some unintended alterations into the symbolic data. Especially if some elements of an input object have been repositioned by large distances, the topology of the output object could well differ from what was intended. Therefore, if altering the symbolic data slightly would result in smaller positional perturbations, we could also take the view that the numeric data is more accurate than the symbolic data. This motivates exploring the consequences of changing the symbolic data, e.g., by subdividing edges and faces, followed by slight positional perturbations of the subdivided elements.

In [9], Milenkovic has explored this approach for operations on planar polygonal regions and for determining the topology of line arrangements in the plane. He presents two techniques,

1. the *data normalization method*, and
2. the *hidden variable method.*

In the data normalization method, we implement Boolean operations on polygonal regions in the plane. Each region is represented by a list of vertices whose coordinates are given as floating point numbers, and a list of edges specified by their vertices. We postulate that no two vertices are closer to each other than some tolerance ϵ, and that, likewise, no vertex is closer to an edge than ϵ. The algorithm begins by altering the input data so as to satisfy these two requirements. Two operations are needed, *vertex*

shifting, and *edge cracking*. These operations are illustrated in Figures 10 and 11.

Vertex shifting merges two vertices that are closer than ϵ, into a single one. Since the representation is based on vertex coordinates, there is no difficulty doing this. Having so identified all vertices that lie close, we next subdivide any edge provided that there is a vertex that lies close to it. If the edge is (u, v), and w lies close to it, then (u, v) is replaced by the two edges (u, w) and (w, v). Thus, new edges and vertices are introduced, thereby modifying the symbolic, topological data. Milenkovic proves that the algorithm terminates and gives bounds on the maximum positional perturbation.

The sequential nature of eliminating near coincidence of vertices and edges in the subdivision method can introduce positional perturbations that are much larger than ϵ. An example for edge cracking is shown in Figure 12. Here, the initial cracking of (u_0, v_0) by the vertices u_1 and v_1 brings the vertices u_2 and v_2 close to the middle segment (u_1, v_1), which is cracked next. This, in turn, introduces further subdivision, so the largest displacement is proportional to $n\epsilon$, where n is the number of vertices. Combined with errors introduced through vertex shifting, Milenkovic shows that the maximum positional perturbation, in the worst case, is proportional to $n\epsilon p$, where n is the total number of vertices, and p is the length of the perimeter.

Milenkovic proposes a second approach in which the positional deviation is globally bounded, called the *hidden variable method*. The algorithm determines how a set of line segments intersect each other within a bounding square, in a topologically consistent manner. Moreover, the coordinates of all intersection points are determined to an accuracy bounded by $\eta = 2^{-k}D$, where D is the diameter of the bounding square and k is the length of the mantissa. This error bound is independent of the number of lines considered.

The method replaces the input lines with *xy-monotonic curves*, that is, curves guaranteed to intersect exactly once with any line parallel to the coordinate axes. In consequence, theorems on line configurations, such as Pascal's theorem mentioned in Section 3 above, need not be true for the topology computed. It derives its name from the fact that these curves are never explicitly constructed, but are known to exist.

One of the basic tools used is a set of procedures that compute, to within predictable accuracy, the intersection of two lines. Here, auxiliary

estimates of the intersection point coordinates make use of other point/line information: For example, if the lines L_1 and L_2 have positive slope and we know two points u and v such that u lies above L_2 and below L_1, whereas v lies below L_2 and above L_1, then this fact entails that the intersection of L_1 and L_2 must lie in a coordinate rectangle whose diagonal is the line segment (u, v). A large part of [9] is devoted to deriving techniques which guarantee this consistency without having to consider all triples of points.

Note that the hidden variable method can be used to implement Boolean operations on polygonal regions, provided we base the numerical part of the representation on line equations rather than on vertex coordinates. Thus, in the worst case, the hidden variable method achieves smaller perturbations than the data normalization method, since the maximum error is independent of the number of input elements.

In [6], Greene and Yao present a method for drawing line segment arrangements on a discrete grid. The objective is to draw the line segments such that the end points and all segment intersections are on the grid points. To this purpose, true intersection points are rounded to the nearest grid points, and the line segments are suitably replaced with polygonal arcs.

Internally to the method, short line segments are considered between true intersection points p and their rounding p'. Such a segment is called a *hook*, and when it intersects an input line segment (u, v), then (u, v) is also split to pass through p'. It is proved that a correct topology can be constructed by first intersecting input line segments, and then intersecting hooks with input line segments. Since the line segments are replaced by polygonal arcs, this approach also alters the symbolic data.

Assuming that the grid points to which we round have integer coordinates, it is proved that the internal precision needed to locate all intersections accurately depends linearly on the lengths of the coordinates of line segment endpoints.

8 Conclusions

We saw in Section 2 that using approximate computation leads to potentially unbounded growth in the number of digits that must be computed. This growth becomes especially acute when geometric operations are iterated, since the numerical errors incurred by each operation may grow multiplica-

tively. The growth of needed precision happens irrespective of the topological structure of objects. However, the topology influences whether the error accumulation leads only to inaccuracy or also to internal inconsistency of the output.

Although we discussed approximate floating point computation, digit growth can also be a problem for exact rational arithmetic, unless countermeasures are built into the algorithms. It is possible to limit the growth of digits. This requires a carefully circumscribed set of permitted geometric operations that must not be left. We saw in Section 5 that useful sets of such operations exist, but the topology of the geometric objects, here polyhedra, did require expensive computations for some very simple tasks such as translating or rotating polyhedra.

In Section 3 we showed that the topology of certain objects may lead to difficult existence problems. Not all object descriptions make sense. We concentrated on purely symbolic descriptions so as to show that this problem is independent of whether we have numeric data or not. However, the main conclusion to draw from the example is that the familiar description of geometric objects using approximate numeric data may contain subtle errors.

The existence problem is the main motivation for drawing a distinction between representation and model, in Section 4. It is clear that we cannot simultaneously represent all geometric models. A simple counting argument shows that. More importantly, we cannot naively assume that a given representation makes sense, even though, based on approximate metric data, the computer is able to give, e.g., a graphical rendering of it. Therefore, we seem to be left with the following choices:

1. Guarantee exact data, as in Section 5.
2. Include reasoning steps into the computation, as in Section 6.
3. Alter the meaning of geometric elements, as in Section 7.

Exact approaches so far have not made a significant impact on geometric modeling systems in practice, due to the perceived inefficiency of the necessary exact arithmetic steps. It may well turn out that the trade-off between robustness and efficiency is much sharper than presently perceived, and that it may require reevaluating exact techniques for incorporation into practical systems.

In the scenario given in Section 6, we stated that the result of certain numeric questions is tested for its sign, and that conclusions are drawn depending on the result being positive, zero, or negative. For instance, the computation might determine the Euclidian distance of a point from a line, and the result would then indicate whether the point is on the positive or negative side of the line, or if it lies on the line. Let us call the latter case a positional *degeneracy*.

Positional degeneracies can be eliminated by perturbing the position of these elements relative to each other, after which the scenario of Section 6 simplifies in that only two outcomes must be considered, namely, whether the result is positive or negative. The challenge in designing a perturbation scheme is to maintain the integrity of the geometric objects, whose elements are perturbed. Thus, the approach is essentially similar to the reasoning paradigm of Section 6. As we demonstrated in Section 3, the difficulty in showing the correctness of a proposed perturbation scheme critically depends on the topological structure of the geometric objects manipulated.

There has been work proposing various such perturbation schemes. The main motivation of this work is to eliminate all degeneracies, which in many cases simplifies the programming effort when implementing geometric operations. Note, however, that positional degeneracies are often intentional in solid modeling, and their systematic elimination would be a mistake, in those applications.

For the SOS method, [4], it is proved that the final perturbations eliminate all existing degeneracies and do not create new ones that were not previously present. In [12], a similar technique is presented. Both schemes require fairly simple geometric input objects. [4] considers points and eliminates collinearity of three and coplanarity of four or more points. [12] takes a more abstract approach. Under the assumption that all numeric computations can be expressed as fixed polynomials in the input parameters, it is shown that consistent perturbations exist that eliminate the possibility of any polynomial evaluating to zero.

Acknowledgements

It is a pleasure to acknowledge the long and fruitful collaboration with John Hopcroft that led to the approach sketched in Sections 4 and 6. Mike Karasick's modeler was the vehicle for trying out some of these ideas, as reported in [7, 8]. Figures 3 – 6 were inspired by similar figures in [11], and

were made with a program implemented by S. Chiang and J. Yu. Victor Milenkovic provided me with a preliminary copy of his thesis of which [9] is an excerpt.

9 References

1. B. Buchberger, G. Collins, and B. Kutzler (1988), "Algebraic Methods for Geometric Reasoning," *Annl. Reviews in Comp. Science*, Vol. 3.

2. D. Dobkin and D. Silver (1988), "Recipes for Geometry and Numerical Analysis, Part 1: An Empirical Study," 4^{th} *ACM Symp. on Comp. Geometry*, Urbana, Ill.

3. G. Collins (1975), "Quantifier Elimination for the Elementary Theory of Real Closed Fields by Cylindrical Algebraic Decomposition," in *Springer Lect. Notes in Comp. Sci. 33*, H. Brakhage, ed.

4. H. Edelsbrunner and E. Mücke (1988), "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms," 4^{th} *ACM Symp. on Comp. Geometry*, Urbana, Ill.

5. G. Golub and C. van Loan (1983), *Matrix Computations*, Johns Hopkins Press, 476p.

6. D. Greene and F. Yao (1986), "Finite Resolution Computational Geometry", 27^{th} IEEE Symp. Found. Comp. Sci., Toronto, 143–152.

7. C. Hoffmann, J. Hopcroft, and M. Karasick (1988), "Robust Set Operations on Polyhedral Solids," Tech. Rept. 723, Comp. Sci., Purdue University.

8. C. Hoffmann, J. Hopcroft, and M. Karasick (1988), "Towards Implementing Robust Geometric Computations," 4^{th} *ACM Symp. on Comp. Geometry*, Urbana, Ill.

9. V. Milenkovic (1986), "Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic," *Intl. Workshop on Geometric Reasoning*, Oxford, England. To appear in *Artificial Intelligence*.

10. J. Owen and A. Rockwood (1987), "Intersection of General Implicit Surfaces," in *Geometric Modeling*, G. Farin, ed., SIAM, Philadelphia.

11. K. Sugihara (1987), "An Approach to Error-Free Solid Modeling," *IMA Summer Program on Robotics*, Inst. Math. and Applic., Univ. of Minnesota.

12. C. Yap (1988), "A Geometric Consistency Theorem for a Symbolic Perturbation Theorem," 4^{th} *ACM Symp. on Comp. Geometry*, Urbana, Ill.

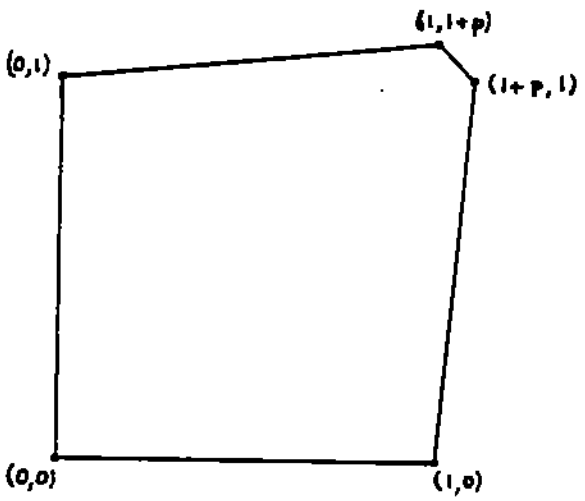

Figure 1: Example Pentagon for *going in* and *going out* Operations

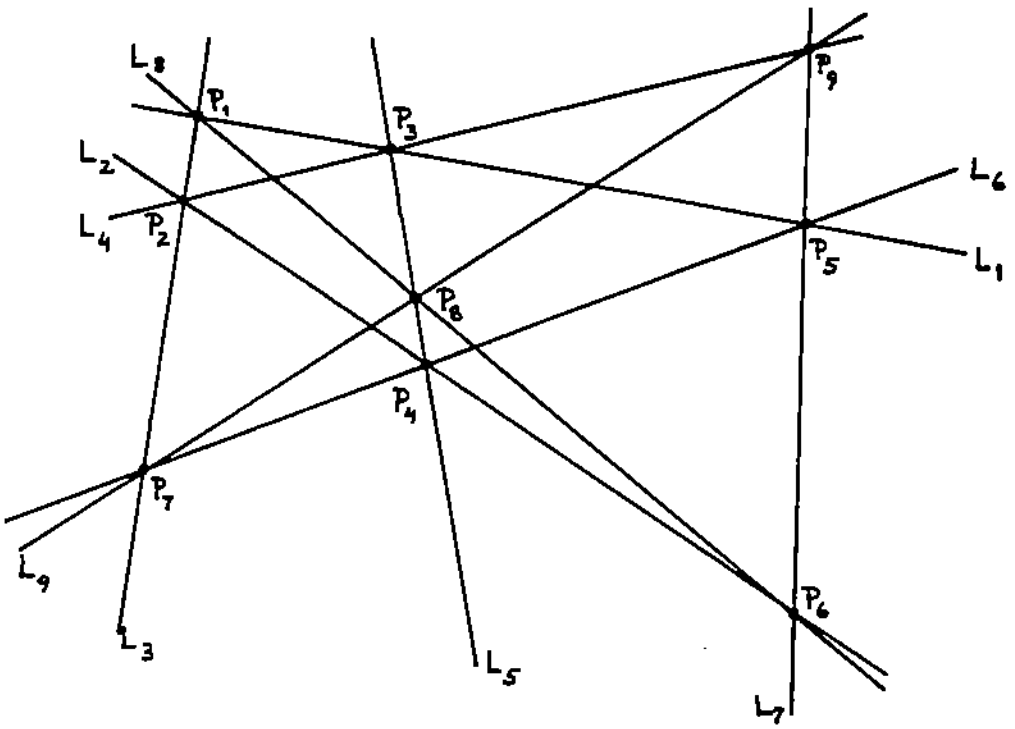

Figure 2: Realizable Point/Line Configuration

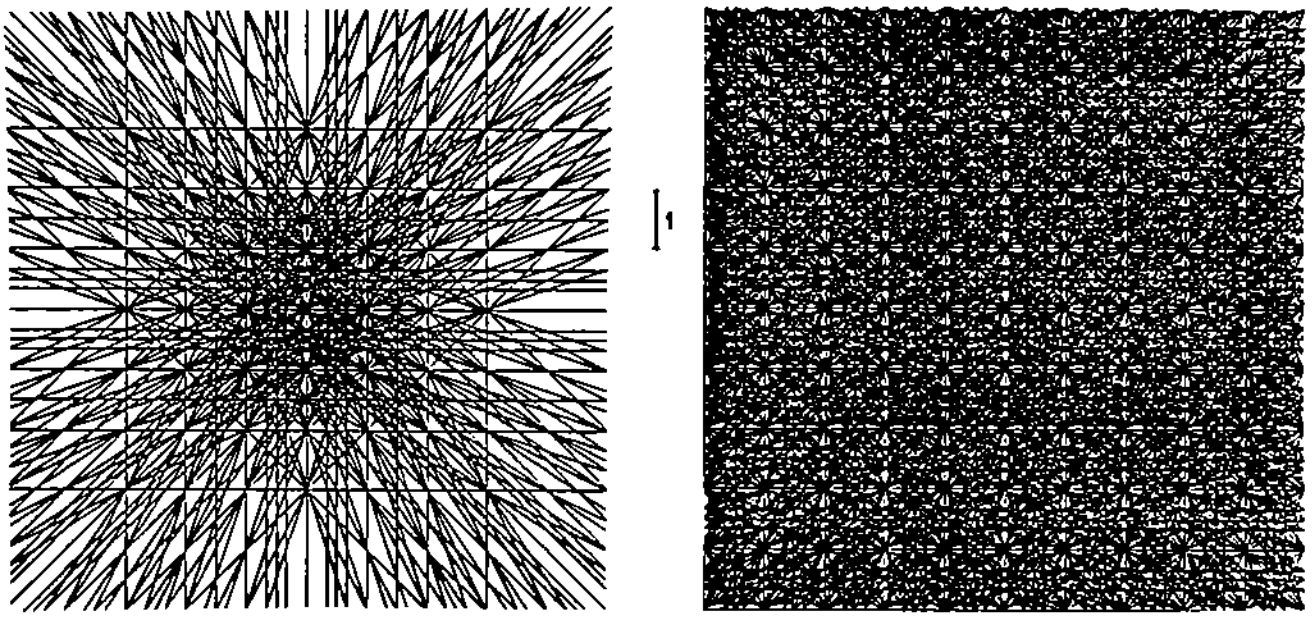

Figure 3: Grid of Lines $ax + by + c = 0$, where $|a|, |b| < 4$. On the left, we have $|c| < 4$, on the right, $|c| < 16$.

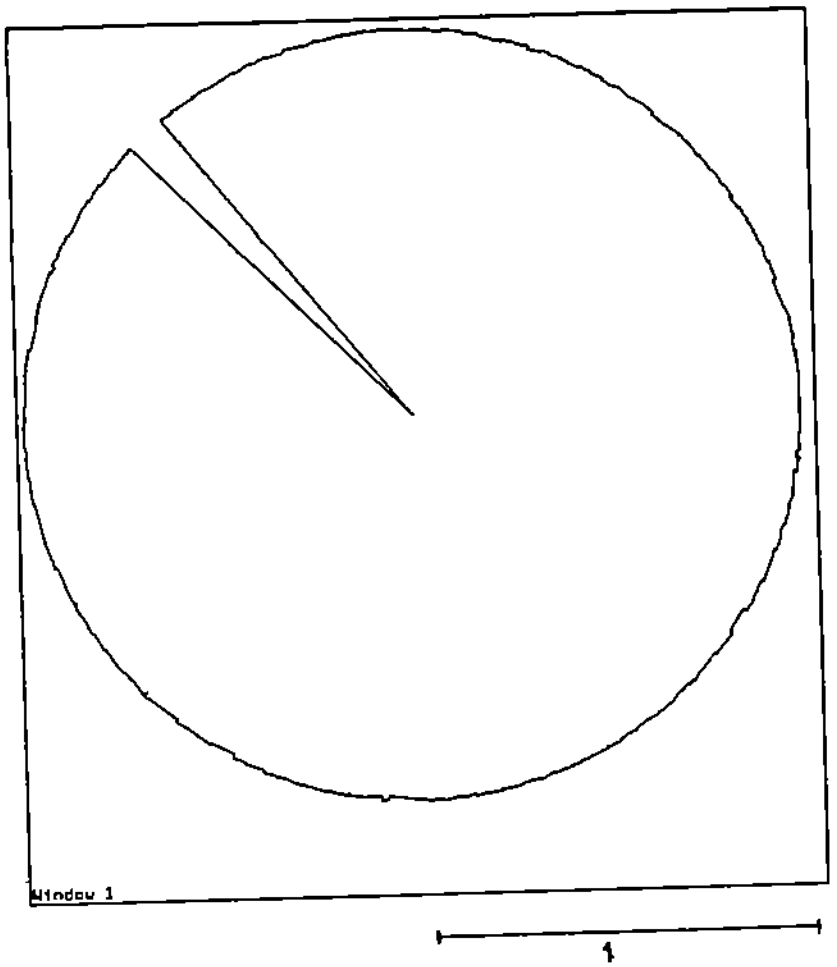

Figure 4: Union of 150 Random Triangles

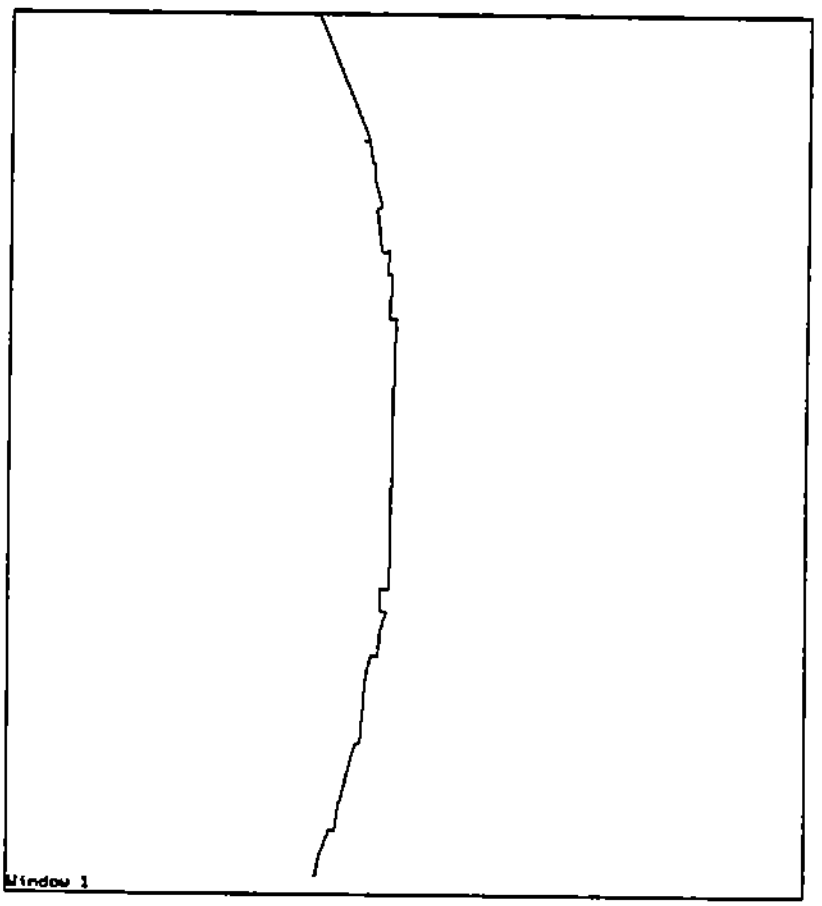

Figure 5: Border, Magnified 3 times

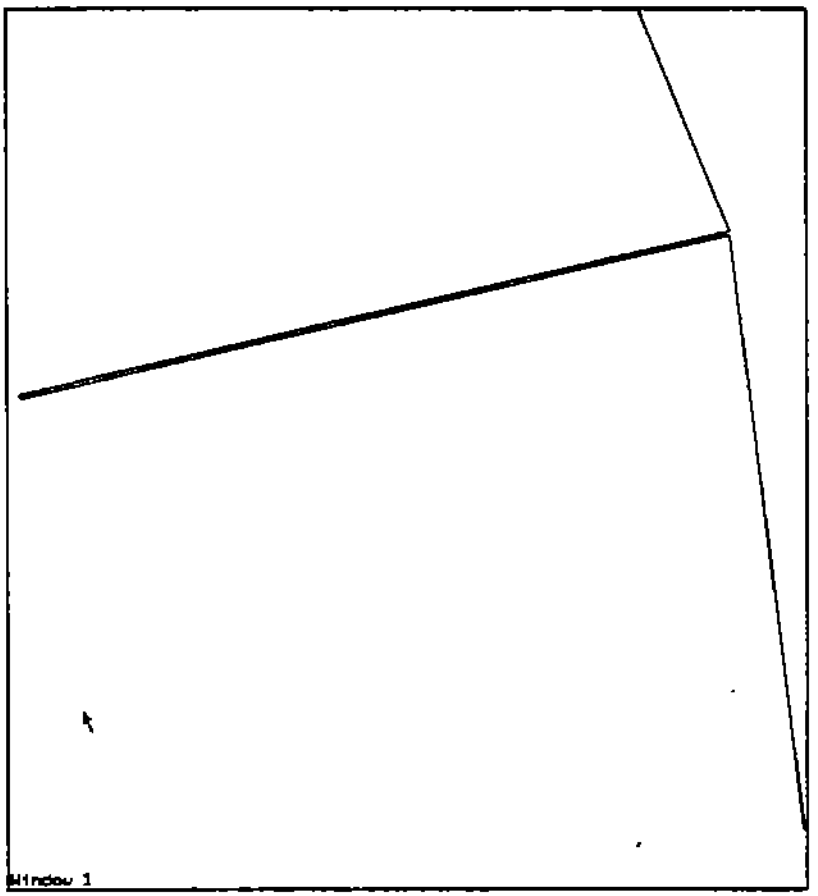

Figure 6: Border, Magnified 500 times

Figure 7: Possible Feature Alteration through Translation or Rotation

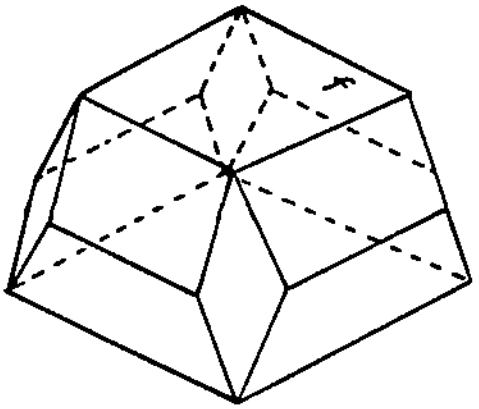

Figure 8: Repositioning the Face f Requires Changing Adjacent Faces

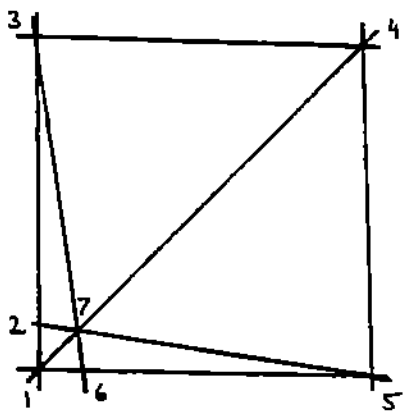

Figure 9: Sequence Dependence of Positional Perturbation

Figure 10: Vertex Shifting

Figure 11: Edge Cracking

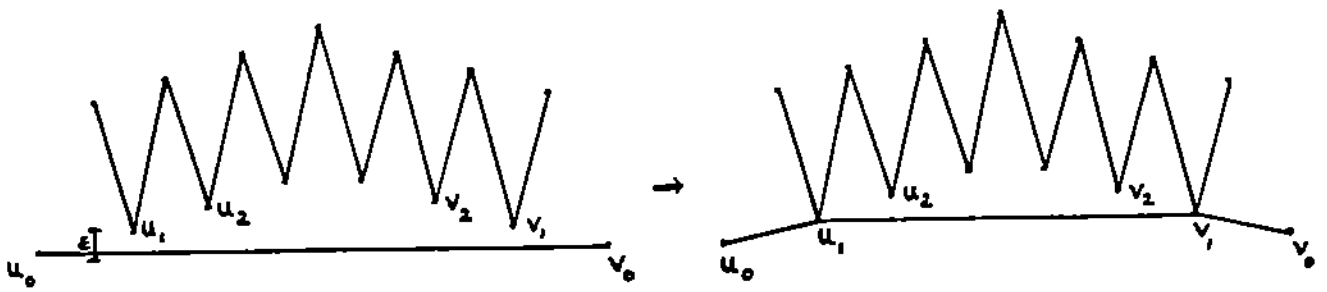

Figure 12: Additive Positional Perturbation in Edge Cracking