Chapter 4

Robust and Error-Free Geometric Operations

Despite much work and great advances in geometric and solid modeling, practical implementations of geometric modeling operations remain error-prone, and the goal of implementing correct, efficient, and robust systems for carrying them out has not yet been attained. This fact seems to originate from an underlying characteristic that sets geometric computations apart from other application areas in computer science and engineering. There is agreement that the problem is serious, but what strategy has the best chance of solving it is not agreed on. The difficulty seems to be rooted in the interaction of approximate numerical and exact symbolic data.

Geometric objects belong conceptually to a *continuous* domain, yet they are almost always analyzed by algorithms doing *discrete* computation. These algorithms typically treat a very large discrete domain — for instance, the set of all representable floating-point numbers — as though it were a continuous domain. This approach may lead to acceptable results in many cases, but it does not work in all situations. In particular, when implementing Boolean operations on solids in B-rep, the problem manifests itself in occasional failures of the implemented algorithm.

We now examine these problems in detail restricting attention to the linear case. Polyhedra and other piecewise linear geometric objects, in three dimensions, consist of points, edges and polygonal faces that are in specific

spatial relationship to one another. The specification of such objects consists of two parts: numerical information, recording vertex coordinates or plane equations; and symbolic data specifying face and edge boundaries, adjacencies, and incidences.

Usually, the numerical data describing a geometric object are given only approximately, using floating-point numbers. In consequence, there may be imprecision that leads to contradictory information about the represented object. For instance, the representation 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 in a single point.

Most implementors are well aware of this imprecision and make allowance for it by suitably relaxing incidence tests. However, this approach does not succeed in all cases, because it is difficult to control the implications of approximate incidence tests. In particular, using limited-precision numbers often has the consequence that the outcome of a numerical computation is sequence-dependent and inaccurate. If the computation is to decide symbolic facts — for example, whether a vertex is incident to a face — the imprecision constitutes incomplete or erroneous information.

Geometric algorithms typically make many such decisions, and all decisions so made must be logically consistent, whether based on correct or incomplete information. This is not always a simple matter. For instance, the same decision can sometimes be determined by different computations that yield contradictory information. Given these possibilities, we must ask what it means for a geometric algorithm to be correct, and for it to deliver an acceptable result for all legitimate inputs.

4.1 Chapter Overview

Since the implementation of geometric algorithms is commonly based on floating-point arithmetic, we show in Section 4.2 some of the consequences that the various numerical inaccuracies can have. Even for line-intersection problems in the plane, floating-point errors may result in geometric contradictions that are usually not anticipated by the algorithm. As we demonstrate, these problems increase in seriousness when we iterate and/or compound geometric operations.

Polyhedral intersection can be implemented correctly when using fixed-precision rational arithmetic. Such implementations can achieve acceptable efficiency provided we correctly anticipate the needed internal precision at which all intermediate calculations are carried out, but they also necessitate reexamining some elementary operations that are commonly taken for granted. This approach is discussed in detail in Section 4.3.

We cannot understand the central core of the robustness problem unless we make a clear distinction between ideal Euclidian geometry and its discrete representation in the computer. In Section 4.4, we discuss this distinction

by defining what constitutes a suitable *model* for a representation, and what might be a good *representation* for an ideal geometric object. In the discussion of these concepts, we will see that a recurrent theme in increasing robustness is to achieve consistent interpretations of noisy numerical data and computations. In this framework, we can distinguish several basic approaches to achieving robustness:

1. Restructure the algorithm such that all interpretations of noisy numerical data and computations are logically independent.

2. Make interdependent logical decisions by respecting the symbolic data exactly, but possibly perturbing the numerical data somewhat.

3. When making interdependent logical decisions, give priority to the numerical data, possibly altering the meaning of the symbolically represented topological problem data.

The first approach assumes that inconsistencies in the representation are probably not fatal unless they lead to contradictory topological conclusions. Thus, this approach tries to devise geometric algorithms in which, for example, an incidence determination is made by only one numerical computation, and all consequent incidences and nonincidences are explicitly recorded at that time. The algorithm of Chapter 3 for performing Boolean operations on polyhedra has been designed based on this concept.

Philosophically speaking, the second approach assumes that the symbolic data have, in principle, an exact representation, and so we should trust them implicitly. This approach is discussed in Sections 4.4.1 and 4.4.4. The technical problems entailed by this approach are how to interpret the numerical data consistently and how to limit the numerical perturbations needed to achieve this consistency.

The third approach, discussed in Section 4.4.5, tries to obtain smaller perturbations by altering the meaning of geometric elements such as lines. Roughly speaking, a line may be replaced by a curve, possibly piecewise linear, with certain properties, such as monotonicity in specific directions or a maximum deviation from a straight line. The technical challenges faced by the third approach include how to use the results of such an algorithm in subsequent geometric computations. Sections 4.5 and 4.6 summarize this material and give references to the literature.

4.2 Floating-Point Arithmetic

Geometric computations customarily use floating-point arithmetic, since it offers both efficiency and flexibility. However, floating-point arithmetic has a number of subtleties that need to be understood. Without a careful accounting for these subtleties, floating-point computations may be the source of unexpected program failure.

4.2.1 Numerical Errors in Floating-Point Arithmetic

A floating-point number consists of an *exponent* and a *mantissa*. Both are fixed-length integers, which implies that they constitute a discrete set of representable rational numbers. Three types of errors must be considered when analyzing the accuracy of floating-point numbers:

1. *Conversion errors*: Since input numbers are usually decimal, whereas the machine operates with binary numbers, we cannot always represent exactly the number desired. For example, the decimal 0.6 is equal to the periodic binary fraction 0.1001 1001... because $3/5 = \sum_{i=1}^{\infty} 9/16^i$.

2. *Roundoff errors*: Since $a \cdot b$ in general requires higher precision to represent exactly than does either a or b, the representation of the product may be inexact in the last represented digit. For example, with five-digit mantissas, the product $0.24665 \cdot 0.63994 = 0.15784\,12010$ would be rounded to 0.15784, thereby incurring an error of approximately $1.2 \cdot 10^{-6}$.

3. *Digit-cancellation errors*: The difference of two nearly equal numbers a and b has fewer significant digits than does either a or b. For instance, the difference of $0.90905 \cdot 10^2$ and $0.90903 \cdot 10^2$ is $0.20000 \cdot 10^{-2}$. Assuming a five-digit mantissa, 0.20000 clearly has only one significant digit, unless a and b happen to be exact numbers.

Moreover, if a geometric shape is expressed by means of a graphical user interface, it may not be possible to specify numerical data precisely.

Since geometric operations usually require extensive numerical calculations, the propagation of these errors is of great concern and influences the accuracy and validity of the geometric operations profoundly. We will demonstrate with several examples that elementary geometric computations are sensitive to such errors, and that elementary geometric conclusions can be invalidated in consequence.

4.2.2 Geometric Failures Due to Floating-Point Arithmetic

We discuss several ways in which a geometric algorithm may fail, assuming that the numerical computation is done using floating-point arithmetic. In the examples, we assume double-precision IEEE standard floating-point computation. The crucial fact is that the computation is carried out with limited-precision arithmetic. We can extend the precision to triple or quadruple precision. This has the effect of *narrowing* the range of inputs for which the geometric algorithm will fail, but it cannot eliminate such failures completely as long as a priori bounds are placed on the precision.

Many geometric questions of incidence can be answered by different sequences of numerical computation. The different sequences of computation are equivalent when exact arithmetic is used. When using floating-point arithmetic, however, these computations may yield different answers. The

following examples illustrate this fact and show how it can affect solid modeling operations.

Incidence Asymmetry

Consider implementing a test of whether two points in the plane are equal. Specifically, assume that the point u is the intersection of the pair of lines (L_1, L_2), and that the point v is the intersection of the lines (L_3, L_4). The line equations are the input to the following algorithm:

1. Compute the coordinates of u.

2. By substitution into the line equations L_3 and L_4, conclude that $u = v$ if both $L_3(u)$ and $L_4(u)$ are smaller than some tolerance.

Intuitively, this algorithm ought to be equivalent to a second version in which the roles of u and v are reversed. We demonstrate that this need not be the case. We assume the following:

1. The intersection (u_x, u_y) of the lines

$$\begin{aligned} a_1x + b_1y + c_1 &= 0 \\ a_2x + b_2y + c_2 &= 0 \end{aligned}$$

   is computed as follows:

$$\begin{aligned} D &= a_1b_2 - a_2b_1 \\ u_x &= (c_2b_1 - c_1b_2)/D \\ u_y &= (a_2c_1 - a_1c_2)/D \end{aligned}$$

2. The point (u_x, u_y) is assumed to lie on the line $ax + by + c = 0$ if the distance is small; that is, if $|au_x + bu_y + c| < \epsilon\sqrt{a^2 + b^2}$.

We assume $\epsilon = 10^{-10}$, a reasonable bound for double precision. We ask whether u and v are incident using two different methods:

1. Compute the coordinates of u; conclude that $u = v$ iff u is on both L_3 and L_4.

2. Compute the coordinates of v; conclude that $u = v$ iff v is on both L_1 and L_2.

The line coefficients follow. Since $2^{-23} \approx 10^{-7}$, they differ from 1 and 0 by amounts that are several orders of magnitude larger than ϵ.

$$\begin{array}{lll} a_1 = -1 & b_1 = 1 & c_1 = 0 \\ a_2 = -(1 + 2^{-23}) & b_2 = 1 - 2^{-23} & c_2 = 2^{-22} \\ a_3 = 1 & b_3 = 0 & c_3 = -(1 + 2^{-15}) \\ a_4 = 0 & b_4 = 1 & c_4 = -(1 + 2^{-15}) \end{array}$$

These coefficients can be represented exactly in double precision. The coordinates of the points are now computed to be

$$\begin{array}{rcll} u & = & L_1 \cap L_2 & = & (1.0, 1.0) \\ v & = & L_3 \cap L_4 & = & (1.000030517578125, 1.000030517578125) \end{array}$$

They are both exact. Moreover, since $(a_i^2 + b_i^2)$ is approximately between 1 and 2, the evaluation of the line equations after substituting point coordinates yields an error that can be compared directly with ϵ. We obtain the values

$$\begin{array}{rcll} L_3(u) & \approx & -3 \cdot 10^{-5} & > \epsilon \\ L_4(u) & \approx & -3 \cdot 10^{-5} & > \epsilon \end{array}$$

from which we conclude that u cannot be incident to v, since it is too far from each of the lines whose intersection is v. But we also obtain

$$\begin{array}{rcll} L_1(v) & = & 0 & < \epsilon \\ L_2(v) & \approx & -7 \cdot 10^{-12} & < \epsilon \end{array}$$

from which we must conclude that v is incident to u, since it lies extremely close to both lines. Therefore, although they ask the same geometric question, the two computations yield contradictory results.

Graphically, the situation is summarized in Figure 4.1. The bound ϵ conceptually defines a narrow band around each line L_i such that every point inside that band is considered to lie on L_i. In consequence, there is a diamond-shaped region enclosing each vertex w, the intersection of the two bands, such that every point inside the diamond region is considered coincident with w. Depending on how the two diamonds intersect, the vertices are considered coincident or not.

How does this asymmetry affect a solid modeler? Recall the problem of intersecting two solids, A and B, in boundary representation, and consider the following conceptual approach:

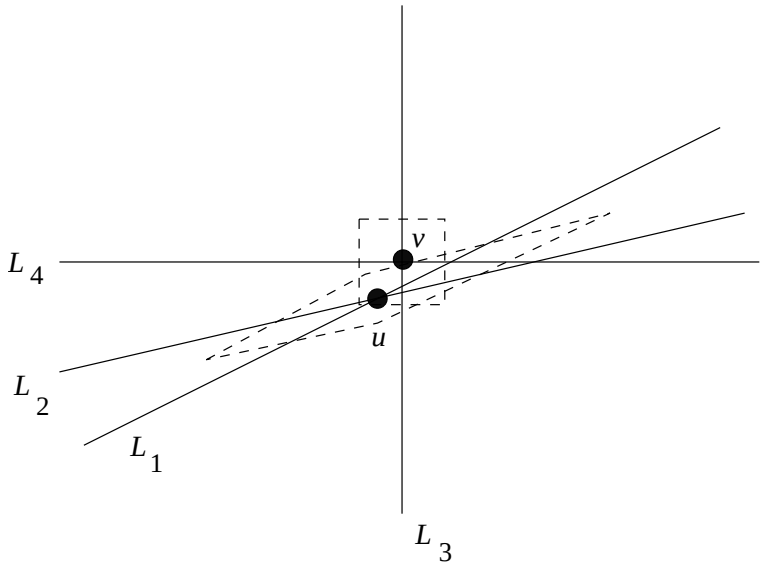

Figure 4.1 Vertex Incidence Regions

1. Mark on the surface of solid A the curves in which the surface of solid B intersects.

2. Reverse the roles of A and B and repeat step 1.

3. Construct the surface of $A \cap B$ by merging the relevant parts of the boundary of A and B.

This algorithm will decide every vertex/vertex incidence with two different computations that are the three-dimensional analogue of the preceding algorithms. Thus, a floating-point implementation will claim contradictory incidences on certain inputs A and B, and the curves marked on both solids will be incompatible in that case, causing the algorithm to fail.

Incidence Intransitivity

Consider introducing symmetry into the point-equality test by asking "is the distance between u and v smaller than a specific threshold?" using the following method:

1. Compute the coordinates of u and of v, by intersecting the respective lines.

2. If the Euclidian distance between u and v is smaller than ϵ, decide $u = v$; otherwise, decide $u \neq v$.

This method is symmetric, but it does not exhibit transitivity. Specifically, we choose the three points u, v, and w such that u is incident to v, v is incident to w, but u is *not* incident to w. We assume that epsilon is 10^{-10}.

$$u = (0,\, 0) \qquad v = (0,\, 0.8 \cdot 10^{-10}) \qquad w = (0,\, 1.6 \cdot 10^{-10})$$

Clearly, the distance between the adjacent pairs is less than ϵ, but the distance between u and w is greater than ϵ.

Assume we intersect two polyhedra, A and B, where one of the edges (u, w) of A is somewhat shorter than 2ϵ. Consider a position of B in which one of its vertices v is approximately in the middle of the edge. Having implemented the symmetric vertex incidence test, we are in the uncomfortable situation that v is incident to two different vertices of A.

Compensating for the intransitivity of point coincidence is difficult. A typical approach is to avoid the problem by requiring that no edge of a polyhedron is shorter than a specific tolerance δ. The magnitude of δ must be related to ϵ. In our example, $\delta > 2\epsilon$ is necessary. However, *every* pair of vertices, whether adjacent or not, must be separated by δ, and possible incidence computations among derived points — for example, edge/face intersections — may necessitate other such minimum-feature-size constraints. The resulting δ will depend on the details of the geometric operation, but the possibility of propagating errors makes it difficult to derive good values.

Topology Violations

The inaccuracy of numerical calculations has other implications for the validity of deduced geometric fact. In particular, when determining how the edges of two polyhedra intersect in three dimensions, we might deduce a configuration such as the one shown in Figure 4.2. When intersecting the edge (u, v) with the top face, we may well conclude that it is sufficiently close to the edge (w, w'), due to large positional perturbations caused by the shallow angle at which (u, v) intersects the top face. Now, when intersecting (u, v) with the side face, a more accurately determined point of intersection can lead us to conclude that (u, v) does not intersect (w, w'). This inconsistency in incidence determination constitutes a violation of the topology that is likely to cause trouble subsequently. Avoiding this problem requires coordinating different incidence computations; for instance, as we did in Chapter 3. As we will see later in this chapter, determining that all decisions have been fully coordinated may be hard.

4.2.3 Line-Intersection Conditioning

As we have seen, the result of a floating-point computation may have considerable error, yet at other times it may be exact. It is not always possible to quantify precisely how floating-point errors propagate in complex sequences

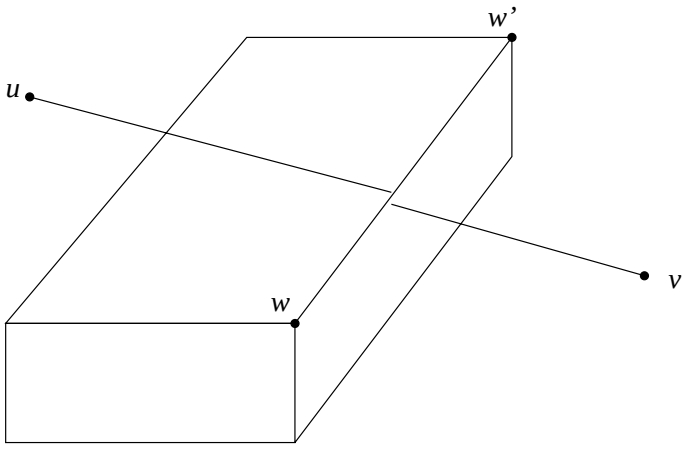

Figure 4.2 Topological Inconsistency When Intersecting Edges

of numerical calculations. However, a sensitivity analysis can be performed that estimates the likely actual error. This analysis is based on the following concept.

We assume that the result of a floating-point calculation is exact. But, instead of being exact for the actual inputs to the computation, the results are exact for some other inputs that are a perturbation of the actual ones. Intuitively, if a small input perturbation yields only a small change in the result, it is reasonable to assume that such a calculation has only a small error. If, on the other hand, small input perturbations — due, for example, to roundoff — may result in large changes of the result, then the computation is likely to have large errors in the result.

More precisely, if an input is changed by ϵ, the output changes by a function $\delta(\epsilon)$. For small values of ϵ, there may exist a constant κ such that $\delta(\epsilon) \approx \kappa\epsilon$. If κ is small, then $\delta(\epsilon)$ is small whenever ϵ is small, and we say that the calculation is *stable* or *well conditioned*. If κ is large, then $\delta(\epsilon)$ is large for small ϵ. In that case, we speak of an *unstable*, or *ill-conditioned* calculation. The number κ is called the *condition number*.

Note that a calculation may be ill conditioned because of inherent sensitivities of the problem to input perturbation, or because of the details of the algorithm used. Although problem ill conditioning is unavoidable, we may be able to circumvent ill-conditioned computations by redesigning the algorithm.

Numerical analysis has developed many stable algorithms. Nevertheless, the propagation of numerical errors cannot always be kept small, since the sensitivity to input perturbations depends in part on the nature of the problem. As an example, we analyze the condition number of the problem of determining the intersection of two lines in the plane. Intuitively, we expect that the intersection can be determined with good accuracy when the lines intersect at nearly a right angle, and we expect poor accuracy for lines that

intersect at angles close to zero or 180°.

Finding the intersection of two lines involves matrix inversion. Given the lines

$$\begin{aligned} a_1x + b_1y + c_1 &= 0 \\ a_2x + b_2y + c_2 &= 0 \end{aligned}$$

we invert the matrix

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

Matrix inversion has been studied extensively in numerical analysis, and the *condition number* of a matrix A measures the sensitivity of the inverse A^{-1} to perturbations in the entries of A. So, the accuracy with which the matrix can be inverted can be estimated by the condition number of A.

The condition number of a matrix can be calculated from the norm of the matrix and its inverse. A *matrix norm* is a function f that maps the matrix A to a nonnegative real number with the following properties:

1. $f(A) = 0$ iff A contains all zeros.
2. If A and B have the same dimension, then $f(A + B) \leq f(A) + f(B)$.
3. For any real number u, $f(uA) = |u|f(A)$.

The *infinity norm* of the $n \times n$ matrix A is

$$\| A \|_\infty = \max_{1 \leq i \leq n} (\sum_{j=1}^{n} |a_{ij}|)$$

The condition number of A is given by

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

We derive the condition number of the matrix A in terms of the angle between the two lines.

Assume that each line equation has been adjusted such that the sum of the squares of the coefficients of x and y is one. That is,

$$a_1^2 + b_1^2 = 1 \qquad \text{and} \qquad a_2^2 + b_2^2 = 1$$

Then the matrix becomes

$$A = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \cos(\alpha+\beta) & -\sin(\alpha+\beta) \end{pmatrix}$$

where α is the angle between the x axis and the first line, and β is the angle between the two lines. The inverse of A is

$$A^{-1} = -\frac{1}{\sin(\beta)} \begin{pmatrix} -\sin(\alpha+\beta) & \sin(\alpha) \\ -\cos(\alpha+\beta) & \cos(\alpha) \end{pmatrix}$$

So, we can estimate $\| A \|_\infty \leq 2$ and $\| A^{-1} \|_\infty \leq 2/\sin(\beta)$, assuming $\beta < 180°$. Therefore,

$$\kappa \leq \frac{4}{\sin(\beta)}$$

This estimate shows that the system is well conditioned for angles β close to $90°$, and is ill conditioned for angles close to $0°$ or $180°$.

Under extremely favorable circumstances, we expect a perturbation of the input coefficients of order 2^{-t}, where t is the mantissa length, due to roundoff in the last representable digit. For small angles β, we have $\sin(\beta) \approx \beta$. Hence, for an intersection angle of $1/2^m$, we expect to lose about $m+2$ binary digits.

We demonstrate the loss of precision due to small perturbation; for example, due to roundoff. Assume that the line intersection (u_x, u_y) is computed as before:

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

We consider 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 $1°$. Moreover, as long as m does not exceed the mantissa length, the coefficients are exact in floating-point representation. With these coefficients, the intersection point $(1, 1)$ is determined without

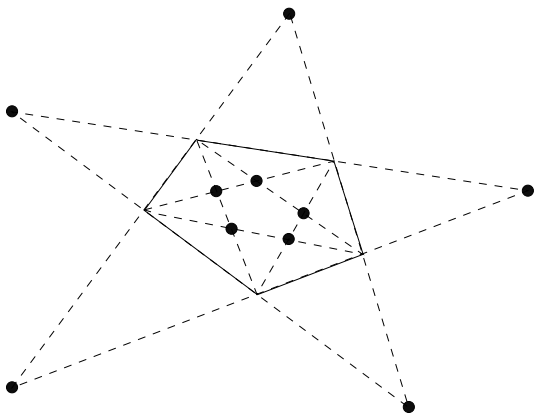

Figure 4.3 *going in* and *going out* Operations

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 1 arc second, resulting in a condition number of 2^{20}. So, we expect to lose about 20 significant binary digits when perturbing the system by machine precision, corresponding roughly to 6 decimals. Double-precision arithmetic carries approximately 16 decimals in a mantissa of 53 bits. The computation for the unperturbed system yields exactly $(1.0, 1.0)$. The perturbed system yields the point $(1 + 3 \cdot 10^{-11}, 1 + 3 \cdot 10^{-11})$ for $t = 53$, in good agreement with predictions.

It is important to remember that the conditioning of line intersection is inherent in the problem, not in the specific algorithm. Some algorithms will do consistently better than others by avoiding, where possible, computations that incur larger perturbations. However, only exact arithmetic will always handle ill-conditioned intersections accurately, provided the input data are not corrupted.

4.2.4 Compound Geometric Operations

A specific difficulty with geometric computation is that we want to subject a geometric object to several operations in sequence. That is, the output of one computation becomes the input to the next. In this situation, precision losses in one operation may accumulate, and may be magnified by subsequent

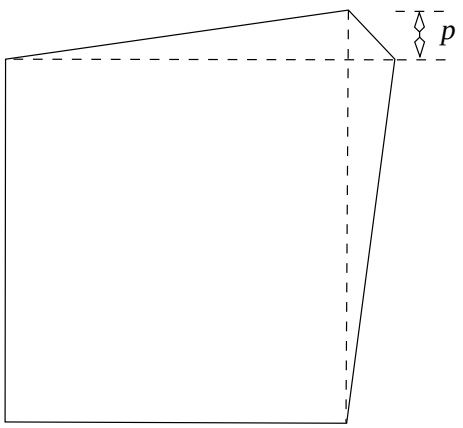

Figure 4.4 Example Pentagon

operations. We consider a simple geometric problem that demonstrates this phenomenon.

Consider a pentagon P in the plane. We draw the five diagonals of P; their intersections define a contained, smaller pentagon Q. Let us call the operation of passing from P to Q *going in*, and write symbolically $Q = in(P)$. Similarly, we extend the five sides of P to their intersections, thus obtaining a larger pentagon Q' that contains P. We call this operation *going out*, and write $Q' = out(P)$. Clearly, $P = out(in(P))$ and $P = in(out(P))$; see Figure 4.3. Beginning with P, let us iterate the *going in* operation m times, obtaining $Q = in^m(P)$, and then compute $P' = out^m(Q)$. Ideally, the coordinates of the vertices of P and of P' should be equal. In practice, they may differ by a large error, even for small values of m.

We take the pentagon with vertices at $(0,0)$, $(1,0)$, $(0,1)$, $(1+p,1)$, and $(1,1+p)$, for small values of p; see also Figure 4.4. Table 4.1 shows the results, with all computations done in double precision. The table demonstrates dramatically that the numerical results from this simple geometric operation

p	$out^2(in^2())$	$out^3(in^3())$	$in^2(out^2())$	$in^3(out^3())$
0.1	$9 \cdot 10^{-14}$	$2 \cdot 10^{-12}$	$1 \cdot 10^{-14}$	$1 \cdot 10^{-13}$
0.01	$8 \cdot 10^{-12}$	$2 \cdot 10^{-9}$	$6 \cdot 10^{-13}$	$7 \cdot 10^{-11}$
0.001	$5 \cdot 10^{-10}$	$2 \cdot 10^{-6}$	$5 \cdot 10^{-8}$	$9 \cdot 10^{-8}$
0.0001	$5 \cdot 10^{-8}$	$1 \cdot 10^{-3}$	$5 \cdot 10^{-5}$	$2 \cdot 10^{-4}$
0.00001	$4 \cdot 10^{-7}$	$7 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	$2 \cdot 10^{-1}$
0.000001	$2 \cdot 10^{-4}$	$7 \cdot 10^{-1}$	$1 \cdot 10^{+2}$	$5 \cdot 10^{+2}$

Table 4.1 Absolute Error for Iterating *going in* and *going out* Operations

can be quite inaccurate.

To understand the reason for the poor accuracy, we must analyze the angles between intersecting lines as a function of p. In the first *going in* operation, no angle is smaller than 45°. In the next *going in* operation, angles as small as $\tan(p)$ arise; in the third operation, angles diminish to approximately $\tan(p^2)$. So, with small values for p, the composite condition number for computing $out^3 in^3(P)$ is proportional to $1/p^3$. Hence, in the column labeled $in^3(out^3())$, we expect the error to grow at least with 10^3, and this is confirmed by the experiments.

4.3 Exact Rational Arithmetic

Given the accuracy problems of floating-point arithmetic, it is tempting to use exact numbers throughout. For geometric objects with linear elements, the most natural choice is to use rational numbers. As we saw in the case of the pentagon problem, however, the iteration of geometric operations can result in an unbounded growth of the digits we need to represent. For this reason, we must limit the numerical precision by defining a grid of representable planes beforehand.

We now study an exact approach to polyhedral intersection based on limited precision rationals. After discussing the grid of representable points, lines, and planes, we explain the details of polyhedral intersection in this framework. To maintain correctness, however, simple operations such as translation and rotation must be considered carefully, since they require approximating points and planes that are not necessarily directly representable. We conclude with a discussion of some of the strengths and weaknesses of this approach.

4.3.1 The Grid of Representable Elements

Consider polyhedra in which the face equations are given numerically, and all other information is symbolic. Thus, vertices are defined as the intersection of three distinct planes, and lines containing edges as the intersection of two distinct planes. A plane equation has the form

$$ax + by + cz + d = 0$$

where a, b, c, and d are integers. We bound the magnitude of the coefficients by requiring that $-L \leq a, b, c \leq +L$, where L might be $2^{48} - 1$. Moreover, d is bounded by the square of L, as $-L^2 \leq d \leq L^2$.

The rationale for bounding the constant coefficient d differently is shown in Figures 4.5 and 4.6, for two dimensions. In Figure 4.5, all coefficients are bounded uniformly. We see that the resulting grid is less uniform than is the grid of Figure 4.6, obtained by bounding d as explained previously. When

computing with projective point coordinates (x, y, z, w),[1] then these bounds become uniform.

Since the projective point (x, y, z, w) corresponds to the affine point $(x/w, y/w.z/w)$, when $w \neq 0$, each point coordinate and each plane coefficient is then an integer of magnitude at most L.

4.3.2 Boolean Operations

We are given two polyhedra with exact plane equations. When we perform regularized Boolean operations on them, the faces bounding the result will be contained in the faces of the input polyhedra. It follows that the set of equations needed to specify the geometric position of the result polyhedron is a subset of the equations of the input polyhedra. Thus, it is always possible to represent the result of Boolean operations provided the input polyhedra are representable, and there will be no growth in the precision of the numerical data.

When examining the algorithm in Chapter 3, we see that the numerical calculations needed to construct the result polyhedron are all reducible to testing whether a point u, given as the intersection of the planes P_1, P_2, and P_3, is above, on, or below a plane P_4. This test can be implemented as follows. Let $P_i = a_i x + b_i y + c_i z + d_i = 0$, and $i = 1, 2, 3, 4$. The coordinates of the intersection $u = (u_x, u_y, u_z)$ can be computed from the following expressions:

$$D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

$$U_x = -\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}$$

$$U_y = -\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}$$

$$U_z = -\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}$$

[1] See Section 5.2 in Chapter 5.

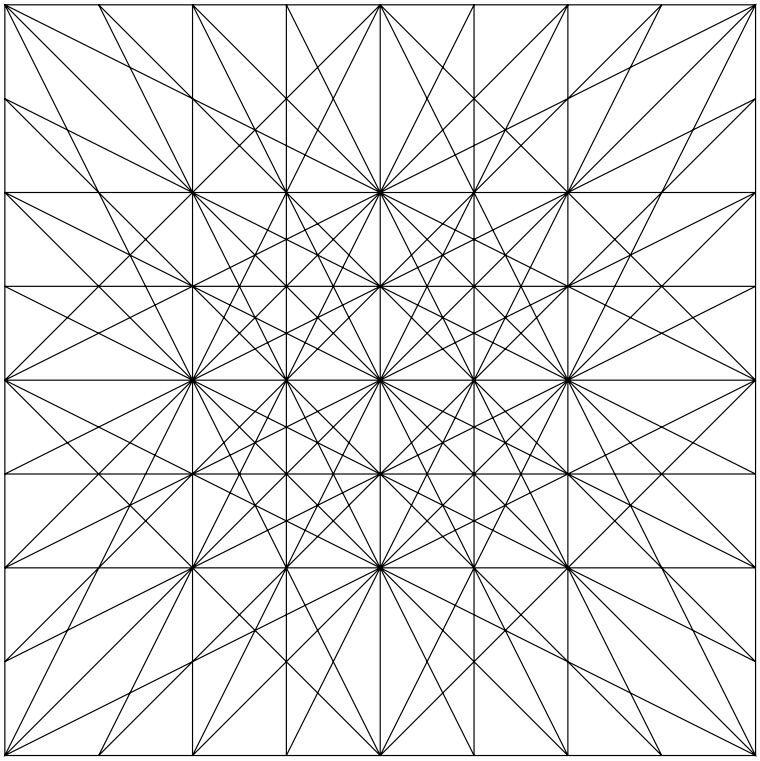

Figure 4.5 Grid of Lines $ax + by + c = 0$, Where $|a|, |b|, |c| < 3$

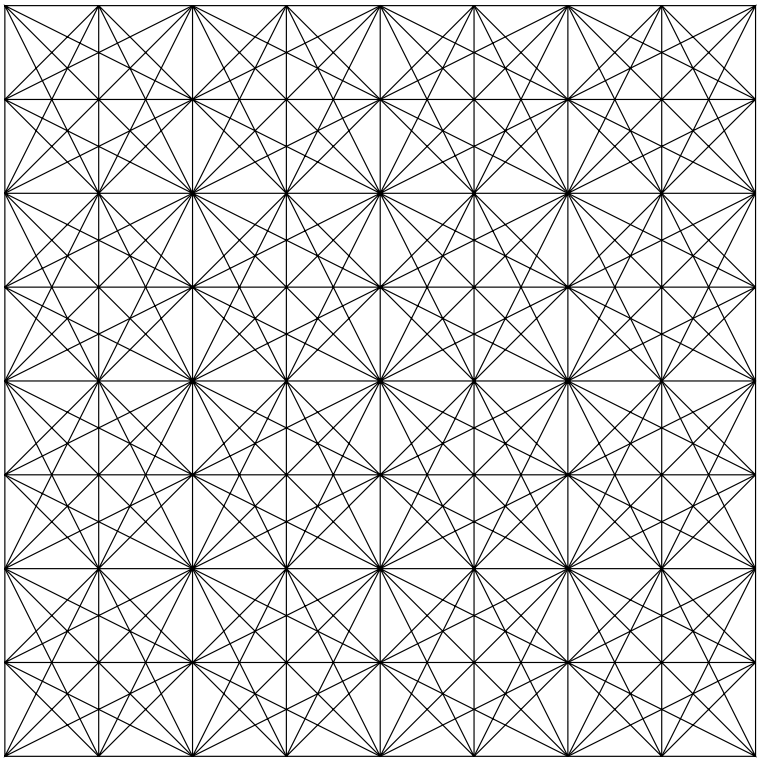

Figure 4.6 Grid of Lines $ax + by + c = 0$, Where $|a|, |b| < 3$, and $|c| < 9$

as

$$\begin{aligned} u_x &= U_x/D \\ u_y &= U_y/D \\ u_z &= U_z/D \end{aligned}$$

Thus, the point u will be above, on, or below the plane P_4 iff the expression

$$a_4 u_x + b_4 u_y + c_4 u_z + d_4$$

is greater than, equal to, or less than zero, respectively. Multiplying by D, this expression becomes

$$a_4 U_x + b_4 U_y + c_4 U_z + d_4 D$$

which is the development of the determinant

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

However, since D may be negative, we must correlate the signs of J and of D. Thus, u is above P_4 iff J and D both have the same sign; it is below P_4 iff J and D have opposite signs. The point u is on P_4 iff $J = 0$.

Example 4.1: Consider the intersection $u = (1, 1, 1)$ of the three planes $P_1 : x - 1 = 0$, $P_2 : y - 1 = 0$, and $P_3 : 1 - z = 0$. Given the plane $P_4 : x + y - z = 0$, we test whether u is above, on, or below P_4. The determinant J evaluates to -1, so u is not on P_4. Moreover, the determinant D is -1, so u is above P_4. $\diamond$

The evaluation of J requires summing products of the form $a_i b_j c_k d_l$. Each product is bounded by L^5, and there are at most 24 such products; hence, the magnitude of J can be bounded by $24L^5$. If the precision bound L is expressed in terms of l binary digits, then this means that performing Boolean operations requires $5l + 5$ binary digits for all intermediate computations.

4.3.3 Rigid Motions

If a grid plane is rotated or translated, it is clear that the resulting plane may not be representable. For example, when rotating the plane $z = 0$ about the x axis by 30°, we obtain the plane $ay + bz = 0$ with a coefficient ratio $a : b$ of $1 : \sqrt{3}$. Thus, this plane cannot be represented with integer coefficients. In consequence, we must investigate ways in which to perform these operations approximately, without violating the integrity of objects. In particular, a plane P that should be moved to a position Q' that is not representable must be moved instead to a nearby plane Q that is representable. We call this process *element rounding*. Throughout, we assume that every plane equation $ax + by + cz + d = 0$ is *reduced*; that is, there is no common factor dividing all four coefficients.

Translation

Consider the translation

$$\begin{aligned} x_1 &= x - t_x \\ y_1 &= y - t_y \\ z_1 &= z - t_z \end{aligned}$$

It maps the plane

$$P\colon\ ax + by + cz + d = 0$$

to the plane

$$P'\colon\ ax_1 + by_1 + cz_1 + e = 0$$

where $e = d + at_x + bt_y + ct_z$. Since P is reduced, P' must also be reduced. Hence, P' is representable iff $|e| \leq L^2$. In that case, the translation is exact. Otherwise, the plane is not representable and must be rounded, as described later.

Rotation

Every rotation can be expressed as a sequence of rotations about the coordinate axes. We therefore restrict the discussion to a single rotation about the z axis. Rotations about the other coordinate axes are analogous. Such a rotation corresponds to a coordinate transformation of the form

$$\begin{aligned} x_1 &= ux - vy \\ y_1 &= vx + uy \end{aligned}$$

where $u^2 + v^2 = 1$; that is, $u = \cos(\alpha)$ and $v = \sin(\alpha)$. Without loss of generality, we assume that $-90° \leq \alpha \leq 90°$.

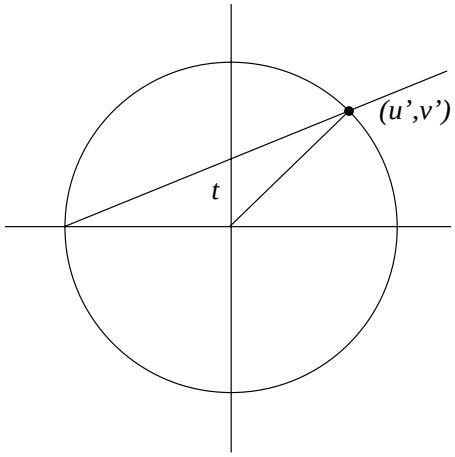

Figure 4.7 Parameter Interpretation of t

Given an angle α, u and v could be irrational. Therefore, we locate a nearby angle α' such that u and v are rational; that is, we seek a rational point (u', v') on the unit circle close to the point (u, v). A simple method for finding rational points on the circle is to use the rational parametric form of the circle, given by

$$\begin{aligned} u' &= \frac{1-t^2}{1+t^2} \\ v' &= \frac{2t}{1+t^2} \end{aligned}$$

where we substitute a rational number for t. The parameter t has a geometric meaning, shown in Figure 4.7, and can be expressed in terms of α as $t = \tan(\alpha/2)$. From the rational approximation m/n of t, we obtain the rational point (u', v'), where

$$\begin{aligned} u' &= \frac{n^2-m^2}{n^2+m^2} \\ v' &= \frac{2nm}{n^2+m^2} \end{aligned}$$

on the unit circle. Using this point, the plane

$$P = ax + by + cz + d = 0$$

is thus transformed into the plane

$$P' = a'x_1 + b'y_1 + cz + d = 0$$

where $a' = au' - bv'$ and $b' = av' + bu'$. After clearing the denominator and reducing the equation, the resulting coefficients might exceed L in magnitude, and, if so, the plane must be rounded.

4.3.4 Rational Approximations

The operations of the previous section necessitate two types of approximations:

1. Given a positive real number w, find a rational number p/q approximating w such that q does not exceed a bound Q.

2. Given a plane equation $ax+by+cz+d = 0$, find a nearby plane equation that is representable; that is, find a plane $a'x + b'y + c'z + d' = 0$ with integer coefficients such that $|a'|, |b'|, |c'| \leq L$, and $|d'| \leq L^2$.

As we shall see, the techniques for approximating a real number can be used for the second problem also.

Rational Approximations of a Real Number

Let w be a positive real number. We wish to approximate w by a rational number p/q such that $q \leq Q$. We exclude exhaustive search, since it takes time proportional to Q, and hence is too slow in practice.

The first method will simply use $q = Q$. Let $p = [wQ]$ be the closest integer to wQ. Then the rational p/Q is an approximation of w. Moreover, since $|wQ - [wq]| \leq 0.5$, the error in this approximation is bounded by $1/(2Q)$. We call this the *naive* approximation method. For example, with $w = 0.123$ and $Q = 100$, we obtain the approximation $12/100 = 3/25$, with a total error of 0.003. Note that the approximation can be constructed in constant time.

The second method uses continued fractions to approximate w. The *continued-fraction representation* of a positive real w is an infinite sequence of integers $(k_0, k_1, k_2, ...)$ such that w is the limit of *convergents* u_i of the form

$$u_0 = k_0, \qquad u_1 = k_0 + \frac{1}{k_1}, \qquad u_2 = k_0 + \cfrac{1}{k_1 + \cfrac{1}{k_2}},$$

$$u_3 = k_0 + \cfrac{1}{k_1 + \cfrac{1}{k_2 + \cfrac{1}{k_3}}}, \qquad \cdots$$

Given a positive real number w, the following algorithm constructs the continued-fraction expansion of w:

1. Set $r_0 = w$ and $k_0 = \lfloor w \rfloor$.
2. Repeat the following for $i = 1, 2, ...,$ until $r_i = k_i$: Set

$$r_i = \frac{1}{r_{i-1} - k_{i-1}} \qquad k_i = \lfloor r_i \rfloor$$

If w is a rational number, then the algorithm terminates eventually with $r_n = k_n$. In this case, we define $k_{n+1} = k_{n+2} = ... = 0$. Otherwise, the algorithm determines an infinite sequence of integers k_i.

The computation of the i$^{\text{th}}$ convergent

$$u_i = k_0 + \cfrac{1}{k_1 + \cfrac{1}{k_2 + \cdots \cfrac{1}{k_{i-1} + \cfrac{1}{k_i}}}}$$

is facilitated by the following recurrence. For $i = 0, 1, 2, ...,$ let $u_i = p_i/q_i$, and define $p_0 = k_0$, $q_0 = 1$, $p_{-1} = 1$, $q_{-1} = 0$. Then

$$u_i = \frac{k_i p_{i-1} + p_{i-2}}{k_i q_{i-1} + q_{i-2}}$$

for $i \geq 1$. The convergents $u_i = p_i/q_i$ are rational approximations of w whose precision is

$$|w - u_i| \leq \frac{1}{q_i^2}$$

Since we bound the denominator of the approximation, it makes sense to speak of the *best* approximant of w. If Q is the bound on the denominator, then p/q is the best approximant if

$$|w - \frac{p}{q}| \leq |w - \frac{r}{s}|$$

for all integers r and s such that $0 < s < Q$. It is known from number theory that the best approximant of w either is a convergent u_i of the continued-fraction expansion of w, or else has the form

$$\frac{k p_{i-1} + p_{i-2}}{k q_{i-1} + q_{i-2}}$$

where $k_i/2 \leq k < k_i$. The latter fraction is called a *quasi-convergent*.

Using these facts, we therefore determine an approximation for w as follows:

1. Compute the continued-fraction expansion, as specified previously.

2. On determination of the next k_i, test whether $q_i = k_i q_{i-1} + q_{i-2}$ is greater than Q. If so, set $n = i - 1$ and stop; otherwise, continue the expansion.

3. Determine the largest k between $k_{n+1}/2$ and k_{n+1} such that $kq_n + q_{n-1} \leq Q$. Choose either u_n or the quasi-convergent using k, depending on which rational is closer to w.

Thus, we obtain a rational approximant p/q satisfying $q \leq Q$. It can be shown that the error of this approximation is bounded by $1/(qQ)$. We call this the *continued-fraction* approximation method.

To analyze the complexity of the continued-fraction method, we consider the number of iterations needed to construct the approximant. Clearly, the slowest growth of the denominator sequence $(q_1, q_2, ...)$ happens when all k_i are 1. In this case, the q_i form the Fibonacci sequence,

$$1, 1, 2, 3, 5, 8, 13, ...$$

Let $\Phi = (1 + \sqrt{5})/2$. From number theory we know that the i^{th} Fibonnaci number is larger than Φ^{i-2}. Hence, the method requires $O(\log(Q))$ steps and determines the best approximant p/q with $q < Q$.

The continued-fraction method is easy to implement. However, the repeated divisions can introduce errors in the approximation. For example, when approximating 0.123, the following sequence of pairs (r_i, k_i) is determined, using double-precision floating-point arithmetic:

$$\begin{array}{lll} k_0 = 0 & & r_0 = 0.123 \\ k_1 = 8 & & r_1 = 0.13008107624663445 \\ k_2 = 7 & & r_2 = 0.687513271369275 \\ k_3 = 1 & & r_3 = 0.45451737681859816 \\ k_4 = 2 & & r_4 = 0.20013590459294717 \\ k_5 = 4 & & r_5 = 0.9966046923658309 \\ k_6 = 1 & & r_6 = 0.00340687500229432343 \\ k_7 = 293 & & r_7 = 0.5241220372356565 \\ \vdots & & \vdots \end{array}$$

Here, r_7 should be 1, since the correct expansion of 0.123 is

$$(0, 8, 7, 1, 2, 4, 1, 0, 0, 0, ...)$$

Thus, it is important to compare the accuracy of the convergents with the original number at each step in the expansion iteration.

Element Rounding

Now consider approximating an arbitrary plane equation by one whose coefficients are bounded by L and L^2, respectively. For notational simplicity, we broaden the problem and approximate

$$a_1x + a_2y + a_3z + a_4 = 0$$

where the a_i are reals and should be approximated with integers bounded separately by $|a_i| \leq L_i$. It is our intention to reduce this problem to separate approximation problems for the coefficients.

The ratio $|a_i|/L_i$ measures by how much the coefficient a_i exceeds its bound. Dividing by the maximum ratio, we obtain a plane equation $b_1x + b_2y + b_3z + b_4 = 0$, in which one coefficient is $|b_k| = L_k$ and the others obey $|b_i| \leq L_i$. Next, we divide by b_k and obtain an equation in which the k$^{\text{th}}$ coefficient is 1.

$$w_1x + w_2y + w_3z + w_4 = 0, \qquad w_k = 1$$

By approximating the w_i with rationals $|p_i/q| \leq |w_i|$, where $q \leq L_k$, we obtain the approximate equation

$$r_1x + r_2y + r_3z + r_4 = 0, \qquad r_k = 1$$

After multiplying with q, all coefficients satisfy the required bounds. Thus, we have reduced the problem of approximating the coefficients simultaneously to a problem of approximating each coefficient separately, albeit with a uniform bound on the denominator.

Example 4.2: Consider the plane equation

$$3.2x + 4.5y + 12.3z + 30 = 0 \tag{4.1}$$

where the coefficients of x, y, and z should be bounded by 3, and the constant term by 9. The maximum ratio is $12.3/3 = 4.1$; hence, we divide by 4.1 and obtain

$$0.7804878x + 1.0975610y + 3z + 7.3170732 = 0$$

Subsequent division by 3 yields the equation

$$0.2601626x + 0.3658537y + z + 2.4390244 = 0$$

So, with a bound $q = 3$, the approximation to equation (4.1) is therefore

$$\frac{1}{3}x + \frac{1}{3}y + z + \frac{7}{3} = 0$$

which is equivalent to

$$x + y + 3z + 7 = 0$$

◇

The naive method for constructing a rational approximant to a real lends itself naturally to the problem of element rounding, since the rational approximations will have uniform denominators.

The continued-fraction method, in contrast, often gives better approximants, but it does not result in uniform denominators. By using the denominator bound $\sqrt[4]{L_k}$, we can circumvent this difficulty, and we can also construct plane-equation approximants.

4.3.5 Object Reconstruction

If a polyhedron has been subjected to a translation or a rotation, some of its elements may have been rounded. In consequence, the integrity of the object may have been violated. To appreciate this problem, consider Figures 4.8 through 4.10, where we have restricted the modeling domain to two dimensions.

Figure 4.8 shows the union of 150 triangles, randomly generated with one vertex on a circle of radius 10^{-4}, and the other two vertices on the unit circle. We observe that the boundary of the resulting object contains many small features, such as the narrow crack shown in magnification in Figure 4.10. When translating or rotating the object, we have no guarantee that in the new position there exist representable grid lines that can bound such a feature. Possibly, then, element rounding may have altered the feature to look as shown in Figure 4.11, so a simple polygon might be changed to one whose edges intersect.

To sidestep this problem, we reduce it to the primitive objects from which all complex polyhedra are built using Boolean operations. That is, when

Figure 4.8 Union of 150 Random Triangles

translating or rotating a complex polyhedron P, we *separately* translate or rotate the primitives from which P has been built, and then reconstruct P from the resulting primitive objects. If the primitives can be translated or rotated without violating their integrity, then this reconstruction approach eliminates the problem for complex polyhedra, albeit with a penalty in efficiency. Moreover, the topology may change slightly, since element rounding may alter somewhat the shape of each primitive.

We now consider the problem of maintaining the topological integrity of primitive objects. Presumably, this problem is simpler, since the topology of primitive objects can be kept very simple. That is, we could restrict the primitives to be parallelepipeds, from which all polyhedra could be constructed in principle by a suitable sequence of regularized Boolean operations. More generally, we postulate that all primitive polyhedra are *trihedral*; that is, exactly three faces are incident at each vertex. Trihedral polyhedra are also called *simple* in the literature. So, when slightly altering the plane equation, trihedral vertices remain trihedral, unless an incident edge of the primitive object is small compared to the positional perturbation of the plane.

Figure 4.9 Border, Magnified Three Times

Assume that the primitive object is a parallelepiped, with the six plane equations

$$\begin{array}{llll} P_{x0}: & -x = 0 & P_{x1}: & x = a \\ P_{y0}: & -y = 0 & P_{y1}: & y = b \\ P_{z0}: & -z = 0 & P_{z1}: & z = c \end{array}$$

Here a, b, and c are the side lengths of the parallelepiped and are positive. They are rational, with a numerator that is less than L^2 and a denominator that is less than L. We note that the vertices on the plane P_{x0} lie below the plane P_{x1}, and that the vertices on the plane P_{x1} lie below the plane

Figure 4.10 Border, Magnified 500 Times

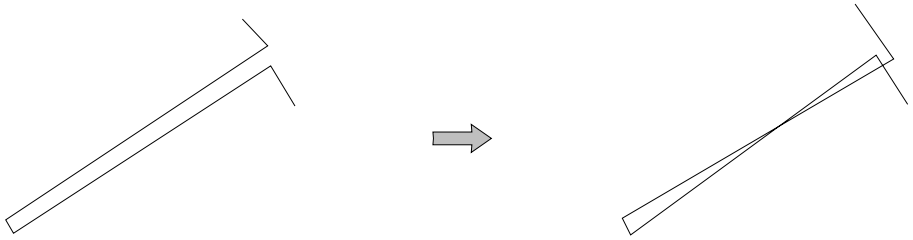

Figure 4.11 Possible Feature Alteration Through Translation or Rotation

P_{x0}. Similarly, the vertices on the plane P_{y0} lie below the plane P_{y1}, and so on. Thus, we have 24 consistency conditions specifying that a certain vertex lies below a certain plane. It is clear that these can be evaluated for the translated or rotated primitive, after element rounding. When satisfied, the new primitive is topologically valid. If one or more condition is violated, then the motion destroys the integrity of the primitive.

4.3.6 Remarks on Using Rational Arithmetic

When using rational arithmetic, it is crucial to control the growth of digits in repeated geometric operations. This is accomplished by requiring that all shape elements be derived from a fixed grid of representable planes. Because translation and rotation may require element rounding, object reconstruction from separately translated or rotated trihedral primitives may be necessary. Moreover, the primitives must satisfy a minimum-feature-separation criterion, so we can be assured that the shape alterations due to element rounding do not invalidate the topology.

Although rational approximation to real numbers is fairly well understood, the process of approximating planes in this context is not fully explored. For example, the quality of the plane approximation depends on which part of it is the final face area. If this area is in the vicinity of the origin, small rotations of the plane can be tolerated, provided the center of rotation is nearby. If the face area is distant, small rotations about the origin can lead to large positional perturbations of the face.

The method has been based on representing the plane equations numerically. Alternatively, we could represent vertex coordinates, placing a bound on the precision. When we do so, a plane on which a set of points lies has coefficients that must be of bounded length. Again, translations and rotations may necessitate element rounding and object reconstruction. Rather than being trihedral, the primitives now must have triangular faces so that

positional perturbations of the vertices, due to element rounding, can be accommodated. In this style of representation, element rounding seems easier and better behaved, since rounding vertex coordinates does not cause large perturbations elsewhere.

As mentioned earlier, element rounding may introduce slight shape alterations. When reconstructing an object, there is no guarantee that two primitives that intersect prior to a rigid motion will also intersect afterwards. In consequence, the reconstructed object may differ in detail from the original object. It is unclear whether the approach can be modified in such a way that the object topology is preserved.

4.4 Representation and Model

When working with floating-point numbers, imprecise numerical results are inevitable. In consequence, internal inconsistencies of the representation of a geometric object are possible. So, we need to elucidate *what* is described by such a representation. Only after we are in possession of a precise geometric meaning of such representations can we address the question whether a geometric algorithm has been correctly implemented. Thus, we introduce the concepts of *representation* and *model*. After these concepts have been explained and a definition of correctness has been given, we discuss several approaches to robust geometric computations and related results that illustrate some of the technical difficulties.

A *representation* is a data structure intended to describe a geometric object, possibly using imprecise arithmetic data. It contains symbolic data describing adjacencies and incidences, and, usually, arithmetic data, such as the plane coefficients for each face. A representation has a *model*, if there exists an object in Euclidian space satisfying the symbolic part of the description precisely. To the numerical data of the representation, there corresponds numerical data of the model. The numerical model data might require infinite-precision numbers.

As an 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 that the faces are square and that opposite faces are parallel. Then any six-sided trihedral polyhedron with quadrilateral faces will be a model, irrespective of the approximate numerical data of the representation that might have been given as vertex coordinates. It is clear that this definition of model is too broad to be useful, so we attempt to capture more accurately the intent of the representation.

Clearly, the intuition of a representation is that the numerical data given are close to the exact data intended. Therefore, it makes sense to compare the exact numerical data of the model with the approximate numerical data of the representation: A model M of a given representation R is *ϵ-close*, if the largest deviation of the numerical data of the representation from the exact model data is not greater than ϵ. This is an absolute error notion that

suffices for our purposes, but it could clearly be replaced by a relative error definition.

A given representation may have exact numerical data. In this case, the representation is its own model, and such a model is called the *natural* model of the representation. To determine whether a given representation possesses a natural model, we verify whether the numerical data, considered to be exact as written, are consistent with the topological data. Of course, this requires sufficient precision in the calculation so as not to lose information. In some cases it is clear that the representation has a natural model. For example, a point set in the plane always has a natural model, since there are no symbolic data to be satisfied. On the other hand, the representation of a polyhedron need not have a natural model.

We can now clarify when a k-ary geometric operation *op* is correctly implemented: The implementation of *op* is *correct*, if for every legitimate input representation R_i there exists a model M_i such that the following are 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 is further refined 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. Here, one wants 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 representation. 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. As we shall demonstrate next with an example, from a mathematical point of view, this appealing intuitive concept is defective.

4.4.1 Models of Purely Symbolic Representations

We consider whether a purely symbolic representation of a geometric object possesses a model. No numerical data are given in the representation; hence, a model exists iff we can assign real numbers to the symbolic coordinates such that the constraint equations implied by the symbolic representation are satisfied. We wish to show that this existence question is nontrivial.

As an example, we consider 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. We consider points and lines in projective 2-space. See also Section 5.2 of Chapter 5.

The triple $[a, b, c]$ symbolizes the line equation $ax + by + cz = 0$, where z is the homogenizing variable. The triple (u, v, w) are the projective point coordinates. Specifying that the point $P = (u, v, w)$ is *incident* to the line $L = [a, b, c]$ means 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 mirror the common requirement of boundary representation schemata that vertices, edges, and faces be distinct, and that they do not intersect except in explicitly specified adjacencies.

Given a symbolic object specification in the preceding 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) are satisfied.

2. All points and lines are distinct and satisfy (D3).

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:

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

This configuration exists in $\mathbf{P}^2$ and is shown in Figure 4.12. However, if the

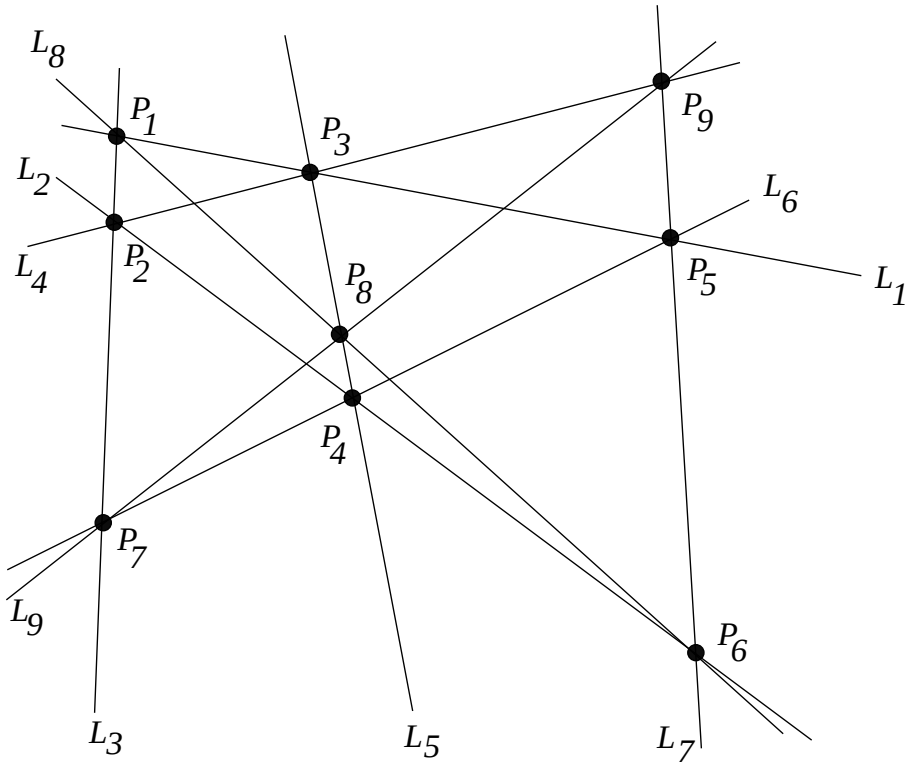

Figure 4.12 Realizable Point/Line Configuration

last incidence constraint, $L_9(P_9)$, is removed, then there is no such configuration in $\mathbf{P}^2$, since that would contradict Pascal's theorem, a fact not easily recognized mechanically.

Consider a conic in which a hexagon has been inscribed. Pascal's theorem states that opposite sides of the hexagon intersect in three points that must be collinear. In Figure 4.12, the conic is degenerate in that it consists of the two lines L_1 and L_2. The hexagon has the vertices P_1 through P_6. The opposite sides $\overline{P_1, P_2}$ and $\overline{P_4, P_5}$ intersect in the point P_7. The other two pairs of opposite hexagon sides intersect in the points P_8 and P_9. By Pascal's theorem, therefore, P_7, P_8, and P_9 must lie on the line L_9.

This example demonstrates that a purely symbolic representation raises existence problems. Were we to base geometric operations on this representation, we would have to verify, for each object, whether it has a model — that is, whether the object exists. There are algorithmic techniques for deciding such questions, but they require potentially excessive symbolic computations and an elaborate theoretical machinery to justify their correctness. Some citations are given at the end of the chapter.

A further difficulty with the purely symbolic representation is that an actual embedding may require irrational coordinates. We demonstrate this fact with the following example. Consider the configuration shown in Figure 4.13, consisting of the nine points $P_1, ..., P_9$ and the nine lines $L_1, ..., L_9$.

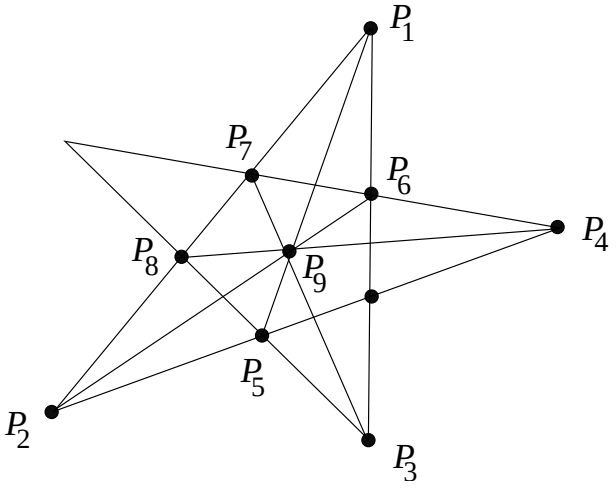

Figure 4.13 Incidence Configuration Requiring Irrational Points

All incidence constraints are easily satisfied, except the incidence of point P_9 with each of the four lines (P_1, P_5), (P_2, P_6), (P_3, P_7), and (P_4, P_8). The configuration is realized by the pentagram some of whose points have irrational coordinates. We assume that it can also be realized with rational point coordinates, and derive a contradiction from this assumption.

If the configuration can be realized with rational coordinates, then every rational projective transformation of it must preserve both the incidences and the rationality of the coordinates. From projective geometry, we know that there exists a nonsingular projective transformation mapping two given quadruples of points into each other, provided that no three points in a quadruple are collinear.[2] So, we may assume without loss of generality that the projective (x, y, w) coordinates of P_1, P_2, P_3, and P_4 are $(0, 0, 1)$, $(1, 0, 1)$, $(1, 1, 1)$, and $(0, 1, 1)$, respectively.

Assume first that neither P_7 nor P_8 is at infinity. Then P_7 may be assigned the coordinates $(a, 0, 1)$ and P_8 the coordinates $(b, 0, 1)$. Note that $a \neq 0$ and $b \neq 0$, since all points must be distinct. So, $P_6 = (a, a, 1 + a)$, and $P_5 = (1, 1 - b, 2 - b)$. The coordinates of P_9 are $(b, b - a, 1 + b - a)$, because P_9 is the intersection of the lines (P_4, P_8) and (P_3, P_7). The point P_9 is also incident to the lines (P_1, P_5) and (P_2, P_6); hence,

$$b + a^2 - 2a = 0$$

[2] In Section 5.5.5 of Chapter 5, such transformations are explicitly constructed. Note that the transformation will be rational when mapping between points with rational coordinates.

$$b^2 - a = 0$$

Elimination of a yields the polynomial $b^4 - 2b^2 + b = 0$ with the roots $b_1 = 0$, $b_2 = 1$, $b_3 = -\frac{1}{2}(\sqrt{5}+1)$, $b_4 = \frac{1}{2}(\sqrt{5}-1)$. The rational roots b_1 and b_2 yield degenerate configurations. Hence, the realization of the nondegenerate configuration requires irrational coordinates, assuming all points are at a finite distance from the origin.

Since the configuration is symmetric, it suffices to consider that P_7 is at infinity. Since P_8 is on the line (P_1, P_7, P_2) and is not equal to P_7, we have therefore the coordinate assignments $P_7 = (1, 0, 0)$ and $P_8 = (b, 0, 1)$. A simple computation shows that now P_3 and P_6 must coincide, as must P_4 and P_9. But then P_9 cannot lie on $(P_2, P_6) = (P_2, P_3)$. In summary, the point coordinates in this configuration cannot all be rational, no matter how the configuration is realized in the plane.

4.4.2 The Role of Decision Making

Assume we implement a geometric algorithm using floating-point arithmetic. We know that we must deal with imprecise numerical data, and that we cannot always be certain that the outcome of some numerical computation allows us to draw correct conclusions. By carefully analyzing the condition number of each calculation, we can establish the following paradigm:

> A numerical computation C is carried out. Subsequent processing depends on making a logical decision based on whether the outcome of the computation is positive, zero, or negative. As long as the magnitude of the result r exceeds a certain threshold $t(C)$, we can make a correct decision based on r. If the magnitude of r is smaller than $t(C)$, then a decision based on r alone is uncertain.

When the decision is uncertain, we could make it arbitrarily; for instance, we could require that a result r of magnitude $|r| < t(C)$ is understood to mean $r = 0$. But such a decision could have consequences for other, later decisions, so we must make each decision in a logically consistent manner.

The paradigm allows us to identify three basic approaches to devising robust geometric algorithms:

1. Restructure all computations such that the logical decisions are independent.

2. Establish consistency of the decisions by symbolic reasoning. The reasoning steps analyze the logical dependencies and assume that the symbolic data are exact as written.

3. Establish consistency, altering the symbolic data as necessary.

4.4.3 Irredundant Decision Making

Inconsistencies among geometrically related logical decisions arise when different numerical computations are performed independently, even though their results are not independent. Therefore, we attempt to restructure the algorithm such that different computations that determine geometrically dependent questions are eliminated. This practical idea is illustrated by the algorithm for regularized intersection of Chapter 3. It requires a careful examination of the possible dependencies present among the numerical steps.

It seems unlikely that the approach is capable of eliminating *all* possibilities for failure. However, since the robustness of an algorithm increases perceptibly even when interdependencies are eliminated only partially, this approach is very attractive in practice.

4.4.4 Preserving Symbolic Data

Dependencies of logical decisions are not always simple to recognize. So, we add symbolic computations to determine logical consequences. How difficult is such symbolic reasoning in specific situations? In the case of intersecting two simple polygons in the plane, it is not at all difficult. However, when intersecting three polygons *simultaneously*, or when intersecting polyhedra, it could be quite hard.

Intersecting Polygons

Consider the problem of intersecting two polygons in the plane. This problem is sufficiently simple that all uncertain numerical results can be decided independently and no symbolic reasoning will be needed to maintain consistency. Nevertheless, the problem is not trivial and, by altering it slightly, we increase its difficulty.

Polygon intersection requires as subroutines numerical calculations to decide whether two edges intersect and whether, in particular, a vertex of one polygon is incident to an edge of the other polygon. The specific logical difficulty in an implementation is that the incidences, as determined by the algorithm, may not be satisfiable in Euclidian geometry; that is, that there need not be models of the input polygons that satisfy the incidences determined in the course of intersection. In view of this, we say that an edge e of polygon A is *overconstrained* if it contains one or more vertices of the polygon B in its interior.[3] For overconstrained edges we need to prove that these additional incidences can be satisfied in Euclidian geometry without sacrificing the fact that the edge is a line segment. Note that vertex/vertex incidences do not create such problems.

As an example, consider the two n-gons shown in Figure 4.14. If the intersection algorithm determines that every vertex of the first n-gon is incident

[3] We refer here to the *relative interior* of the edge; that is, to the edge without its end points.

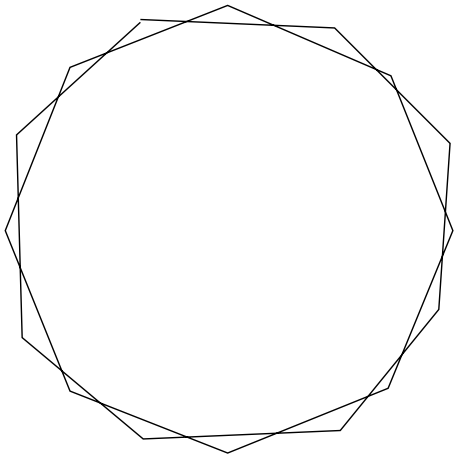

Figure 4.14 Intersecting Two n-Gons

to an edge of the second n-gon, and vice versa, then no two polygons exist in Euclidian geometry that satisfy every one of these incidences:

Proposition
Given two polygons A and B in the Euclidian plane, there is at least one edge that is not overconstrained.

Proof
Assume that the two polygons have m and n vertices, respectively. A vertex of polygon A cannot overconstrain two or more edges of B; hence, if every edge is overconstrained, then $m = n$. Consider the convex hull of the vertices of the two polygons. Then at least one vertex, say of A, must be a vertex of the convex hull, and this vertex cannot constrain any edge of B. Hence, there is at least one edge in B that is not overconstrained. □

Now, if at least one edge is not overconstrained, then we can construct two model polygons[4] that satisfy all incidences that may have been postulated by the implementation. This is done by a placement strategy that constructs the model polygons in a specific sequence. The intuition is as follows: If an edge is not overconstrained, then it can be placed last. Thus, we remove all such edges, with their vertices, obtaining a set of polygonal arcs. Some of the removed edges have constrained other edges, so now there will be new unconstrained edges, since we removed edges along with their endpoints. These edges are removed next, and the procedure is repeated until no edges remain. Thereafter, the edges are reconstructed in Euclidian space and are placed such that the required incidences are satisfied. Since, at the time of placing, an edge is not overconstrained, there is no difficulty preserving the linearity of all edges. Thus, we can construct model polygons that satisfy

[4]Polygons need not be simple; that is, the boundary is allowed to self-intersect.

all required incidences, and therefore can also obtain a model that is the intersection of these input models.

With these observations in mind, we implement polygon intersection in the expected manner, but require that any two vertices of a given polygon are no closer than 3ϵ to each other, and that the minimum distance from any vertex to another edge is also at least 3ϵ. If any vertex of A is closer than ϵ to an edge or vertex of B, then we decide that the distance is zero. If every edge is overconstrained, then we arbitrarily undo one such incidence. This must create an unconstrained edge somewhere, and hence permits the construction of suitable models establishing correctness.

After two polygon representations have been so intersected, the resulting representation need not satisfy the minimum separation bound on vertices and edges. Thus, a postprocessor may be needed that restores the minimum-feature-separation condition. Postprocessing may require obliterating short edges; that is, it affects the symbolic data as well.

A key factor in the correctness of this procedure is that we do not consider additional constraints such as the possible collinearity of different polygon edges. Were we to do so, then the approach would fail because now a single vertex of B could overconstrain two or more edges of A.

A similar difficulty arises when intersecting three polygons *simultaneously*, for a single vertex of one polygon may simultaneously overconstrain an edge in each of the other two polygons. In particular, we can obtain configurations such as the one shown in Figure 4.12 from superimposing three polygons. We conclude that the simultaneous intersection of three or more polygons is more difficult, since it allows us to create configurations that are the subject of theorems in projective geometry. That is, in such problems, different incidence decisions are logically interdependent in possibly complex ways.

Remarks on Preserving Symbolic Data

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 different edges. These constraints can be introduced, for example, by considering the *simultaneous* intersection of three or more polygons.

Polyhedral intersection in three dimensions is considerably more difficult, since it no longer is evident how to reposition a face consistently to satisfy incidence decisions. As an example, consider the polyhedron A shown in Figure 4.15. 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 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 — that is,

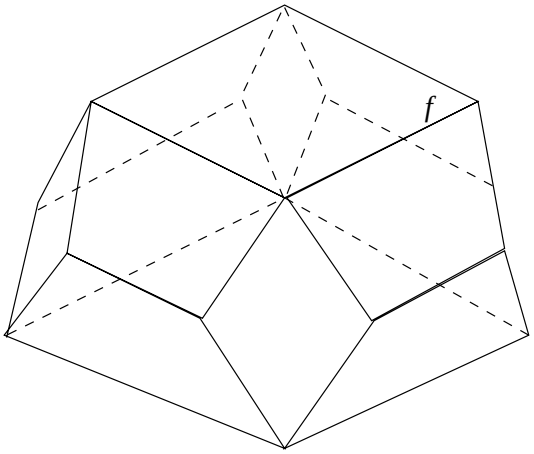

Figure 4.15 Repositioning the Face f Requires Changing Adjacent Faces

whether there exists a model of the altered polyhedron representation — is not immediately clear.

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 a 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, when proving correctness of an implementation in this framework, we have to show that we can satisfy the incidence constraints introduced during the course of the computation, by consistently repositioning the elements of the input models. In all likelihood, this repositioning is sequential, for proof purposes; for instance, as in the polygon intersection algorithm. But the repositioning sequence affects the final position of the vertices, edges, and faces. For example, consider the configuration shown in Figure 4.16. Here, positioning vertices in the order 1, 3, 4, 5, 2, 7, 6 is much more favorable than is 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 based on the inputs, but also that the specific sequence leads to small positional perturbations.

4.4.5 Altering the Symbolic Data

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

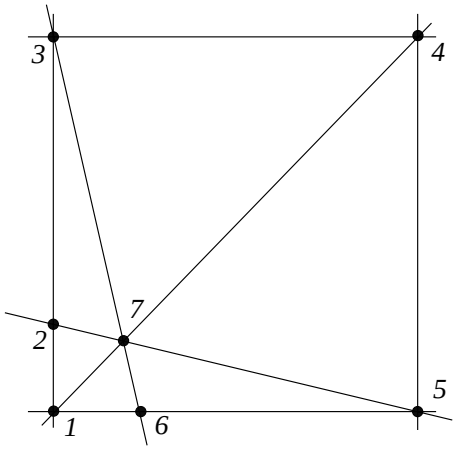

Figure 4.16 Sequence Dependence of Positional Perturbation

data in the representation ought to be correct as given. However, objects are often constructed from other objects by geometric operations, so there is the chance that the implementation has introduced some unintended alterations into the symbolic data. Especially if some elements of an input object have been repositioned over 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 numerical data are more accurate than are the symbolic data. This motivates exploring the consequences of changing the symbolic data; for example, by subdividing edges and faces, followed by slight positional perturbations of the subdivided elements.

For polygonal regions in the plane, we can base such an approach on the concept of normalizing the input data. We postulate that no two vertices are closer than some tolerance ϵ, and that, likewise, no vertex is closer to an edge than ϵ. The algorithm alters the input data to satisfy these two requirements. Two operations are needed, *vertex shifting* and *edge cracking*, illustrated in Figures 4.17 and 4.18.

Vertex shifting merges two vertices that are closer than ϵ into a single one. There is no difficulty doing this if we base the representation on vertex coordinates. Having so identified all vertices that lie close, we next apply edge cracking and 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.

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 4.19. Here, the initial cracking of (u_0, v_0) by the vertices u_1 and v_1 brings the

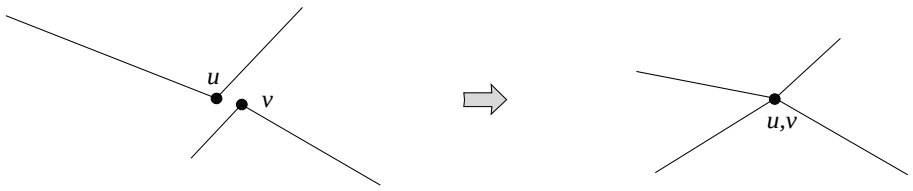

Figure 4.17 Vertex Shifting

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 vertex displacement, in this case, is proportional to $n\epsilon$, where n is the number of vertices.

4.5 Discussion

We have discussed several competing approaches for dealing with the accuracy and robustness problems in geometric computation. Roughly speaking, they fall into one of four categories:

1. Guarantee exact data by using bounded rational arithmetic, or exact algebraic numbers.

2. Ameliorate the problem by restructuring the algorithm to limit the redundancies among the numerical computations performed.

3. Include reasoning steps in the computation, but try to satisfy the symbolic input data exactly.

4. Alter the meaning of the geometric elements; that is, modify the symbolic data in an attempt to minimize positional perturbations.

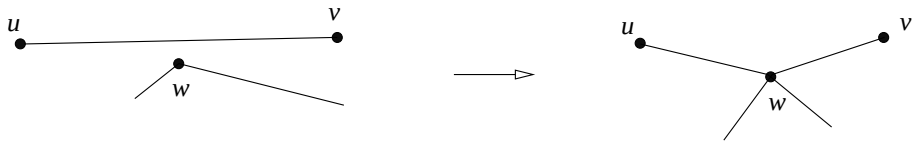

Figure 4.18 Edge Cracking

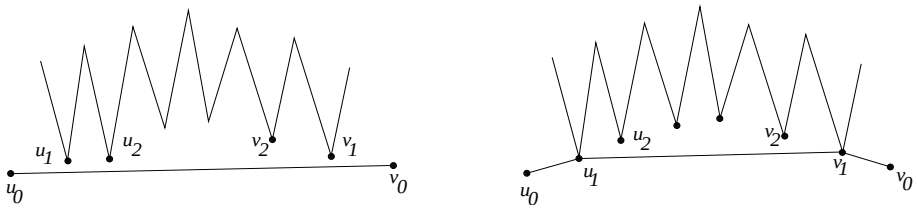

Figure 4.19 Additive Positional Perturbation in Edge Cracking

So far, exact approaches have not had a significant impact on geometric modeling systems in practice, due to the perceived inefficiency of implementing exact arithmetic computations. It may well turn out, however, that the tradeoff between robustness and efficiency is so exacting that we may have to reevaluate our demands for efficiency.

For linear geometric objects, the bounded-precision-rational arithmetic approach offers both accuracy and a measure of efficiency. However, the need to reconstruct P for a rigid motion and the fact that this may alter the topology of P are limitations. It is not clear how conveniently the approach can generalize to geometric objects with nonlinear elements. Specifically, the intersection of planes with integer coefficients is a rational point, but the intersection of quadric surfaces with integer coefficients need not be rational. Already in two dimensions, the intersection of the circle $x^2 + y^2 - 1 = 0$ with the line $x - y = 0$ is irrational. Once the range of coefficients is fixed, it is clear that there must be a fixed minimum distance between any two representable distinct points. However, estimates of the needed precision to separate them are unfavorable.

In Section 4.4.1, we showed that the topology of certain objects may lead to existential problems, since not all object descriptions make sense. We concentrated on purely symbolic descriptions to show that this problem is independent of whether or not we have numerical data. An important conclusion to draw from the example is that the familiar description of geometric objects using inexact numerical data may contain subtle errors.

The existence problem is the main motivation for drawing a distinction between representation and model. It is clear that we cannot simultaneously represent all geometric models; a simple counting argument shows that. More important, we cannot naively assume that a given representation makes sense, even though, based on approximate metric data, the computer is able to give, for instance, a graphical rendering of it that appears to be meaningful.

In practice, heuristics are employed that ameliorate but do not eliminate the robustness problem. Conceptually, we view these heuristics as attempts to reduce the logical interdependence of decisions that are based on numerical computations. For example, when intersecting polyhedra, the asymmetry of the algorithm presented in Chapter 3 ensures that many incidence questions are not asked in two different ways. Rather, once incidence or nonincidence has been decided on, this fact is used to subdivide the faces of polyhedron A and of polyhedron B simultaneously. Other heuristics include structuring the computations such that numerical *input* data are used where possible, rather than *derived* numerical data being used. For example, when testing edge/edge intersection, the computations are expressed in terms of the plane equations involved, and do not use a derived parametric representation of the two edges.

A strict adherence to the topological data seems to necessitate symbolic reasoning. In general, such reasoning can be expensive; see also the next section. It is possible that some geometric operations of interest to solid modeling do not require expensive reasoning, but this topic is relatively unexplored.

4.6 Notes and References

Condition numbers and related concepts for analyzing the sensitivity of floating-point algorithms are standard techniques in numerical analysis. In the case of linear geometric objects, books on matrix techniques, such as Golub and van Loan (1983), are useful. The pentagon example of Section 4.2.4 was proposed by Dobkin and Silver (1988), who advocate extending the precision of the mantissas dynamically during the computations as required. Similar numerical examples can be generated with matrices. First, multiply A and its inverse each m times with itself. Then, multiply the resulting matrices with each other, and compare the product to the identity matrix.

The material on exact rational arithmetic is from a lecture K. Sugihara gave in 1987 at the IMA Summer Program on Robotics at the University of Minnesota. This material is appearing now; see Sugihara and Iri (1988 and 1989). Tighter bounds can be given on the internal precision needed to decide incidence. Using Hadamard's inequality, Sugihara bounds the magnitude of the determinant J by $16L^5$. For sharp lower bounds on the minimum separation distance see Yu (1991). Several element-rounding techniques are discussed in Sugihara (1987). For plane rounding, Sugihara also considers the *basis-reduction method* — see, for example, Lovász (1986) — which usually gives better results than does the continued-fraction method. However, both methods ignore the interaction between coefficient perturbation and the locality of the final face on the plane.

Karasick, Lieber, and Nackman (1989) experiment with adaptive techniques for reducing the cost of rational arithmetic without bounding the precision of the rationals involved. Given a determinant $|A|$ with rational

entries whose sign must be determined, they try to transform $|A|$ to another determinant $|A'|$ whose entries are also rationals but have numerators and denominators of smaller precision. Under certain conditions, the determinant $|A'|$ has the same sign as $|A|$ and can be found quickly. In that case, the evaluation of $|A'|$ is cheaper. If the conditions are not met, then $|A|$ must be evaluated and no savings are realized. Karasick, Lieber, and Nackman report that a Delaunay triangulation algorithm with rational arithmetic can be sped up by several orders of magnitude using this approach.

The notion of representation and model, and the approach to designing robust implementations that preserve the symbolic data, were developed in Hoffmann, Hopcroft, and Karasick (1988). The paper notes the connection between Pascal's theorem and the simultaneous intersection of three polygons, and an expanded version discusses the line configuration requiring irrational coordinates. The idea of limiting the interdependence of numerical computation is from Hoffmann, Hopcroft, and Karasick (1987).

The problem of whether a specific point/line configuration can be realized in Euclidian geometry can be investigated using oriented matroid theory. Bokowski and Sturmfels (1986) and Bokowski, Richter, and Sturmfels (1989) give an algorithm solving the problem. See also Bokowski and Sturmfels (1989).

A variation of the embedding problem arises in geometric theorem proving: Given an embedded configuration, we typically ask whether certain additional incidences are satisfied. Different techniques for solving such problems can be found in Chou (1988), in Kapur (1986), and in Kutzler (1988). All algorithms for geometric theorem proving may require exponential running times on planar configurations composed from circles and lines. As Hong (1986) shows, the symbolic computations arising in this context can be replaced by equivalent numerical computations carried out at sufficiently high precision. However, in the case of configurations consisting of circles and lines, exponentially many digits may be needed. See also Section 7.6 in Chapter 7 for a Gröbner basis approach to geometric theorem proving.

The method of Section 4.4.5 is called *data-normalization*; see Milenkovic (1988). Milenkovic proposes a second paradigm for altering the symbolic data, called the *hidden-variable* method. In it, each line is replaced by an x, y*-monotonic curve* having the property that the curve does not intersect a line parallel to the coordinate axes more than once and is close to the line within some global bound. The method is based on careful computations of line-intersection points. Whenever a new intersection point is determined, consistency calculations are performed that may refine other line-intersection points, so that a topologically consistent function can be constructed that assigns to each point/line pair one of the labels `on`, `above`, or `below`, with the obvious meaning.

A related approach to finite-precision geometric computation is presented in Greene and Yao (1986). Considering an integer grid of representable points and a set of line segments whose endpoints are representable, they break line

segments to reposition intersections on representable points. Theorems are given that prove termination and consistency of the procedure.

Recall that the result of certain numerical computations must be compared with zero, and that conclusions are drawn depending on whether the result is positive, zero, or negative. For instance, the computation might determine the Euclidian distance of a point from a plane, and the result would then indicate whether the point is on the positive or negative side of the plane, or that it lies on the plane. Let us call the zero case a *positional degeneracy*. Positional degeneracies can be eliminated by perturbing the coordinates of the elements relative to each other, after which we would need to consider only two possible outcomes; namely, whether the result is positive or negative.

Most geometric algorithms simplify substantially when positional degeneracies are eliminated from consideration, and this is a major motivation of research on this subject. Edelsbrunner and Mücke (1988) propose the "SOS method," and prove that the final perturbations eliminate all existing degeneracies while not creating new ones. Yap (1988) presents a similar technique. Both schemes require fairly simple geometric input objects and exact arithmetic. Edelsbrunner and Mücke (1988) consider points and eliminate collinearity of three and coplanarity of four or more points. Yap (1988) takes a more abstract approach. Under the assumption that all numerical computations can be expressed as fixed polynomials in the input parameters, he shows that consistent perturbations exist that eliminate the possibility of any polynomial evaluating to zero.

It would seem that perturbation schemes not only pay off in that geometric algorithms become simpler, but that they also could increase robustness. However, two difficulties must be addressed:

1. We must ascertain that a perturbation is permissible. In the context of solid modeling, a positional degeneracy such as the coincidence of two face planes may be intentional. A perturbation of the relative position of the planes could be contrary to the designer's intentions, and hence would be inappropriate.

2. The challenge in designing a perturbation scheme is to maintain the integrity of the geometric objects whose elements have been so perturbed. As we saw in the example of point/line incidences, this can be a difficult problem.

The first difficulty is problem-dependent, and there are applications in which perturbation makes sense. For instance, perturbations seem to be appropriate when eliminating hidden lines during graphic display operations; see Sugihara (1989).

The second difficulty is technical in nature. For example, Yap's method would need to be extended such that the zeros of a certain subset of the polynomials are preserved. This subset would contain polynomials that express constraints of incidence, coplanarity, and so on. Such extensions have

not been incorporated thus far. However, geometric theorem proving could provide the necessary tools fairly conveniently.