# Polyhedral modeling with multiprecision integer arithmetic ⋆

Steven Fortune

*Bell Laboratories, Murray Hill, NJ, 07974, USA*

We describe a polyhedral modeler that uses software extended-precision integer arithmetic to guarantee numerical reliability. By careful design, the performance of the modeler is not much different from the performance that a floating-point modeler might have. The modeler performs Boolean set operations exactly; to prevent growth of coordinate bit-length, affine transformations require coordinate rounding and hence are approximate. A new algorithm for reconstructing polyhedral incidence information after rounding is given.

*Key words:* polyhedral modeling, exact arithmetic, adaptive-precision arithmetic, robustness, geometric algorithms, numerical reliability, winding number

## 1 Introduction

We describe a three-dimensional boundary-based polyhedral modeler that uses extended-precision software integer arithmetic for geometric predicates. Integer arithmetic guarantees that geometric predicates are reliable and hence that the modeler will not fail because of numerical error. The additional performance overhead of the integer-arithmetic modeler, relative to a modeler implemented with floating-point arithmetic, is minimal. For a 'generic' intersection, there is essentially no overhead. For a contrived hard intersection, the performance cost is less than twice what a floating-point modeler might require (though the floating-point modeler might well fail because of numerical error [15]).

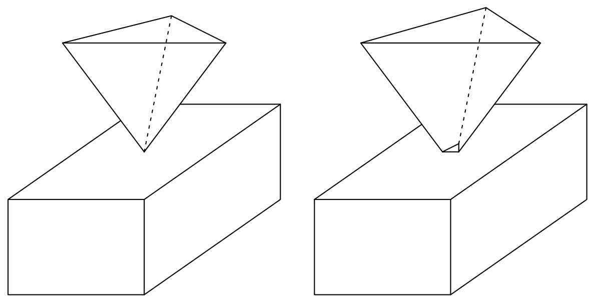
⋆ An earlier version of this paper appeared in the Third Symposium on Solid Modeling and Applications [7].

A principal property of the modeler is a bound on the bit-length of coordinate data; this implies a fixed, relatively small bound on the bit-length of arithmetic required to evaluate geometric predicates. The bit-length bound restricts the class of representable polyhedra. Boolean operations such as intersection or union do not increase the bit-length of coordinate information; such operations are exact. Other operations, such as affine transformations, would increase coordinate bit-length if implemented directly. Instead, coordinates are rounded to the bit-length bound, and the result is only an approximation to the exact answer. Polyhedral incidence information may have to be reconstructed after coordinate rounding; a reasonably simple reconstruction algorithm is described below.

The bit-length required to evaluate geometric predicates, while relatively small, exceeds the native hardware bit-length of most computers. The modeler uses a two-level adaptive-precision strategy to evaluate geometric predicates [9]. A predicate is determined by the sign of an arithmetic expression. The expression is evaluated first in floating-point arithmetic; if the magnitude of the expression is larger than an error bound, than the sign of the value is correct. If not, the expression is evaluated exactly with specially-tuned software extended-precision arithmetic. The error bound can be determined statically, so the error bound check is cheap.

Besides guaranteeing numerical reliability, exact integer arithmetic allows the use of symbolic perturbation [3,24]. We describe a variant scheme that perturbs polyhedral face planes. This symbolic perturbation scheme simplifies the implementation of the modeler by eliminating many special cases. In addition, it allows a simple manifold representation of any polyhedral set, that is, any bounded set that can be obtained from open or closed halfspaces by a finite number of unions and intersections. This includes sets with dangling edges or faces, open or closed boundaries, etc.

**Other work.** Considerable research effort has been directed at improving the numerical reliability of geometric algorithms; for surveys see [5,13,14,20]. One approach is to analyze the effect of rounding errors that result from floating-point arithmetic [6,26,28]. A second approach is to use software exact arithmetic for the evaluation of geometric predicates; to reduce performance cost, various researchers have suggested adaptive-precision arithmetic [9,16,17]. The specific problem of constructing a reliable polygonal or polyhedral modeler has been considered both in floating-point arithmetic [4,12,15] and exact arithmetic [1,16,22,27]; much of the latter work is discussed in more detail below.



(a) (b)

Fig. 1. (a) nonmanifold; (b) manifold representation.

## 2 Algorithm design

A polyhedral modeler provides boolean set operations on polyhedral solids, as well as affine transformations such as rotation and translation. The design described here uses many ideas from Sugihara and Iri [27], though the rounding algorithm is different.

The discussion in this section assumes general familiarity with polyhedral modelers [13,19]; only differences resulting from the use of integer arithmetic are highlighted. The discussion is relatively informal; the appendix contains technical details.

### *2.1 Coordinate data*

Face plane coordinates are the primary geometric representation of a polyhedral solid. A boolean operation on solids creates no new primary geometric data, since a boolean operation does not introduce any new face planes[27]. Vertices are defined as the intersection of three planes.

Plane coefficients have bit-length bound $(B, D)$: for the coefficient tuple $(a, b, c, d)$, representing the plane $\{(x, y, z) : ax+by+cz+d=0\}$, $a, b, c$ are integers of bit-length $B$ (i.e. have magnitude less than $2^B$) and $d$ has bit-length $B+D$. An arbitrary plane $\pi$ within distance $2^D$ of the origin can be approximated by a plane $\pi'$ of bit-length $(B, D)$ so that the unit normal vectors of $\pi$ and $\pi'$ differ by at most about $2^{-B}$, and so that the distances of $\pi$ and $\pi'$ from the origin differ by at most about $2^{-B}$. If $\pi$ is further from the origin, up to distance $2^{B+D}$, an approximating plane can still be found, with approximation error increasing with distance from the origin.

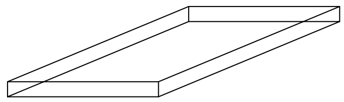


Fig. 2. Manifold representation of rectangle in 3d.

### *2.2 Symbolic perturbation*

Suppose that face plane $\pi_i$ has coefficients $(a_i, b_i, c_i, d_i)$. The computation proceeds as if the plane had coefficients

$$(a_i, b_i, c_i, d_i + \phi_i \epsilon^i)$$

where the *perturbation direction* $\phi_i$ is 1 or $-1$, the exponent $i$ is different for each face, and $\epsilon > 0$ can be viewed informally as an arbitrarily small real or formally as an infinitesimal [23] (see the appendix). Clearly the perturbation translates the plane parallel to itself. This perturbation prevents four planes from meeting at a unique point and three planes from meeting at a common line (though other degeneracies, e.g. four planes all parallel to a common line, are still possible). As a consequence, the modeler can assume that all solids are trihedral, that is, exactly three faces are incident to each vertex.

An advantage of symbolic perturbation is that an arbitrary polyhedral solid can be represented with a simple manifold data structure. By convention, the perturbation direction is chosen so that face planes are perturbed outwards, i.e. to the exterior of the solid. For example, the nonmanifold solid in figure 1(a) has the manifold representation in figure 1(b); the point of contact between cube and tetrahedron has the combinatorial representation of a triangle. A two-dimensional rectangle sitting in three dimensions could have the combinatorial representation of a cube; in figure 2, the top and bottom faces have the same plane equation but are symbolically perturbed in opposite directions. A line segment or point could be obtained in a similar fashion. Outward perturbation yields closed polyhedral sets; open polyhedral sets could be obtained by perturbing face planes inward.

### *2.3 Geometric primitives*

The orientation test on planes, together with a few simpler tests, suffice for the implementation of the polyhedral modeler. Stolfi [25] discusses plane orientation in the context of oriented projective geometry.

The orientation of four planes $\pi_i$, $\pi_j$, $\pi_k$, $\pi_l$ is $\mathbf{sign}(M)$, where

$$M = \begin{vmatrix} a_i & b_i & c_i & d_i \\ a_j & b_j & c_j & d_j \\ a_k & b_k & c_k & d_k \\ a_l & b_l & c_l & d_l \end{vmatrix}$$

($\mathbf{sign}$ has value $-1$, 0, or 1). Suppose $\pi_i$, $\pi_j$, $\pi_k$ meet at a unique point $p$. In homogeneous coordinates,

$$p = (-M_1, M_2, -M_3, M_4)\mathbf{sign}(M_4)$$

where $M_i$ is the determinant of the $3 \times 3$ matrix obtained by deleting the last row and $i$th column of the matrix above. Here we are requiring that the final 'weight' coordinate of $p$ be positive; this implies that the sign of the dot-product $\pi_l \cdot p$ determines whether $p$ is in the positive halfspace, on, or in the negative halfspace of $\pi_l$. Notice that

$$\pi_l \cdot p = M\mathbf{sign}(M_4)$$

so the dot-product is just an alternate form of the orientation test.

If all planes have bit-length $(B, D)$, then each coordinate of the first three columns of $M$ has bit-length $B$, and the last column has bit-length $B{+}D$. Hence the coordinates of $p$ have bit-length about $3B{+}D$ except for the last, which has bit-length about $3B$. Both $M$ and the dot-product $\pi_l \cdot p$ have bit-length about $4B{+}D$.

The orientation test for symbolically perturbed planes requires substitution of perturbed plane coefficients into the orientation determinant. The result when expanded is the original determinant plus a polynomial in $\epsilon$ (see figure 3). The sign of the result is determined by the sign of the first nonzero coefficient, with coefficients taken in order of increasing power of $\epsilon$. Thus evaluating a symbolically perturbed determinant requires the evaluation of the unperturbed determinant, and if it is zero, up to four $3 \times 3$ subdeterminants. A nonzero value for the sign is obtained as long as one of the $3 \times 3$ subdeterminants is nonzero, equivalently, as long as three of the four planes meet in a unique point. If all subdeterminants are zero, then the result is still zero, and the degeneracy is unresolved.

The symbolic perturbation scheme does not resolve all geometric degeneracies. An extended scheme might resolve all degeneracies, but such an extension is

$$\phi_i\epsilon^i \begin{vmatrix} a_j & b_j & c_j \\ a_k & b_k & c_k \\ a_l & b_l & c_l \end{vmatrix} - \phi_j\epsilon^j \begin{vmatrix} a_i & b_i & c_i \\ a_k & b_k & c_k \\ a_l & b_l & c_l \end{vmatrix} +$$

$$\phi_k\epsilon^k \begin{vmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_l & b_l & c_l \end{vmatrix} - \phi_l\epsilon^l \begin{vmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{vmatrix}$$

Fig. 3. $\epsilon$-terms in perturbed orientation determinant.

not necessary. The precondition of the orientation test, that three of the four planes meet in a unique point, is always satisfied by high-level algorithmic properties of the modeler. For example, the classification of a vertex with respect to a plane is only relevant if the vertex is the unique intersection of three planes.

The polyhedral modeler does not require the geometric primitive that compares vertex coordinates. Such a comparison would require cross multiplication, since coordinates are homogeneous, and hence arithmetic bit-length about $6B+D$, more than the orientation primitive. The extension of symbolic perturbation to the comparison primitive would also be more complex.

### *2.4 Face-face intersection*

Face-face intersection is a fundamental subproblem in the implementation of boolean operations [13]; it illustrates the use of the orientation test and symbolic perturbation. Given two faces $f$ and $g$, the edges on each face induced by intersection with the other face must be determined. This requires finding each edge of face $f$ that intersects the face plane $\pi_g$ of $g$, and symmetrically for $g$. Each plane/edge intersection defines a point along the line $l = \pi_f \cap \pi_g$. The new edges on faces $f$ and $g$ can be deduced by traversing these points in sorted order along $l$. In figure 4, there are two edges of $f$ that cross $\pi_g$ and two edges of $g$ that cross $\pi_f$; each such edge defines a point on $l$. Edge $e$ lies in the common interior of faces $f$ and $g$ and is added to both faces.

To sort edge/plane intersection points, observe that each such point is the intersection of $l = \pi_f \cap \pi_g$ and a plane of another face incident to $f$ or $g$. Ordering two such points along $l$ can be expressed as an orientation test on four planes. Symbolic perturbation guarantees that any two such intersection points are distinct. Thus the absence of "geometric degeneracies" implies that

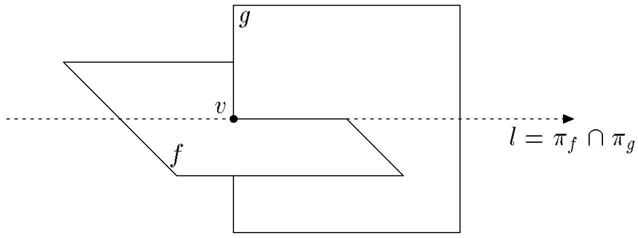



Fig. 4. Face-face intersection. Vertex $v$ may already exist as a result of a previous face-face intersection between $f$ and a face $h$ incident to $g$.

the sort of all edge/plane intersection points is straightforward.

To determine the edges of $f$ that intersect $\pi_g$, each face cycle of $f$ is traversed in order. A dot-product can be used to classify each vertex of the face cycle with respect to plane $\pi_g$; if the endpoints of an edge have different classifications, then the edge crosses $\pi_g$.

The classification of a vertex $v$ of $f$ with respect to plane $\pi_g$ is more complex. As long as $\pi_g$ is not one of planes defining $v$, symbolic perturbation guarantees that $v$ is never reported as on $\pi_g$. However, a previous face-face intersection between $f$ and a face $h$ incident to $g$ might have left an edge on $f$ with endpoint $v$ defined by $\pi_g$ (see figure 4). This "symbolic degeneracy" must be treated as a special case: the code that records edges on faces must discover vertex $v$, as it might be the endpoint of an edge to be added.

## 3 Polyhedral rounding

An affine transformation on a polyhedral solid is effected by multiplying all face plane coefficients by a $4 \times 4$ transformation matrix. Since coefficients are integers, the transformation matrix must have integer entries, and the bit-length of plane coefficients increases by about the bit-length of matrix entries. To preserve the $(B, D)$ bound, plane coefficients must be rounded. Rounding slightly perturbs face planes, and hence may invalidate combinatorial incidence information.

Sugihara and Iri [27] suggest that every polyhedral solid be defined by a sequence of constructive solid geometry (CSG) operations on primitive solids. Each primitive should be "well-conditioned" in the sense that its combinatorial information should not be affected by small perturbation of face planes. To round a polyhedral solid that is the result of a transformation, Sugihara and Iri suggest applying the transformation to the primitive solids, rounding them, and then reapplying the CSG operations. The resulting solid is guaranteed to have consistent combinatorial information, since CSG operations are always valid.

### *3.1 Generalized polyhedra*

A different approach eliminates the need for a CSG definition of solids. The definition of polyhedra is extended to allow a polyhedron to self-intersect. A polyhedron $P$ can then be rounded to another polyhedron $P''$ in two steps. First, the face planes of $P$ are rounded to short bit-length; this results in a consistent but possibly self-intersecting polyhedron $P'$. Then $P''$ is defined as the polyhedron bounding the core of $P'$; the *core* of $P'$ is the set of all points of positive winding number with respect to $P'$. The core computation requires simplification, described below. $P''$ does not intersect itself, has short bit-length plane coefficients, and approximates $P$. For illustration, the approach is described first for polygons in two dimensions, and then for polyhedra in three dimensions.

**Polygons.** A *polygon* of a set of edge cycles, each with an associated traversal direction. Figure 5(a) depicts a polygon $P$, with traversal direction shown by the arrow. Perturbing the edges of $P$ might result in the self-intersecting polygon $P'$ in figure 5(b).

Recall that the winding number $w(q, Q)$ of point $q$ with respect to polygon $Q$ is the sum of the number of oriented intersections of a ray leaving $q$ with the edges of $Q$. A ray-edge intersection counts $+1$ if the traversal direction crosses the ray from right to left, and $-1$ if left to right. A polygon is *simple* if it has no self-intersections; the winding number of any point in the interior of a simple edge cycle is $+1$ or $-1$ as the cycle is oriented counterclockwise or clockwise. Polygon $P$ in figure 5(a) is simple and every interior point has winding number $+1$. Polygon $P'$ in figure 5(b) has a "tail" bounding a region of winding number $-1$ and overlapping "teeth" bounding a region of winding number $+2$.

*Simplification* replaces a possibly self-intersecting polygon $Q$ with a set $N(Q)$ of nesting simple edge cycles so that for any point $q$, $w(q, N(Q)) = w(q, Q)$. (The winding number with respect to a set is just the sum of the winding numbers with respect to the elements of the set; a set of polygons is *nesting* if each pair of polygons either have disjoint interiors or the interior of one is contained in the interior of the other.) Figure 5(c) depicts the set $N(P')$. Simplification is accomplished by the application of the transformation in figure 6 to every pair of intersecting edges; the transformation splits each edge in two and then locally redirects edge cycles as shown.

The polygon bounding the core of $Q$ consists of just the polygonal cycles in $N(Q)$ that separate a region of winding number 0 from a region of winding number $+1$. In figure 5(c), the two such cycles are the big $V$-shaped cycle and

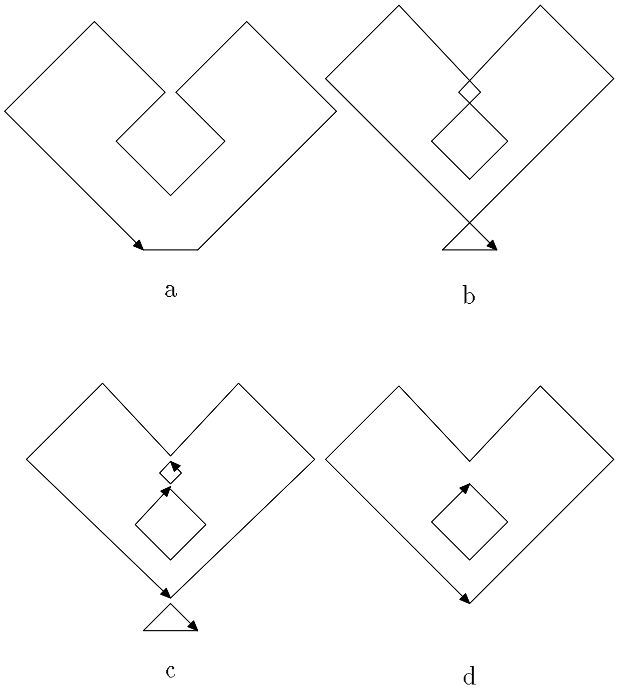



Fig. 5. (a) original polygon $P$; (b) polygon $P'$ obtained by perturbing edges; (c) decomposition $N(P')$ into nested simple cycles; (d) the core $P''$, i.e. the polygon bounding points of positive winding number.

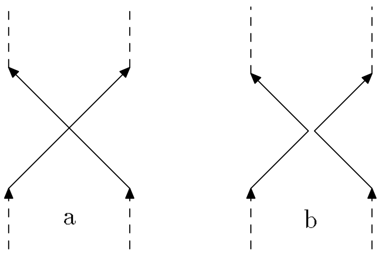



Fig. 6. Intersecting edges (a) are removed by splitting each edge and redirecting cycles (b).

the larger diamond-shaped cycle; these form the polygon $P''$ bounding the core of $P'$.

**Polyhedra.** Simplification-based rounding is similar for polyhedra, though technically more involved. The approach is sketched informally here; the appendix contains more details.

The definition of a polyhedron requires a careful separation of combinatorial incidence structure from geometric embedding. Combinatorially, a polyhedron is a set of symbolic faces. Each symbolic face is a set of face cycles, where each face cycle is a cyclic list of symbolic faces. The polyhedron $P$ shown as a wireframe in figure 7(a) has the face cycles shown in figure 7(b); explicitly, for example, face $F2$ is the cycle ($F1$ $F4$ $F5$ $F3$). The geometric embedding is determined by assigning a plane equation to each symbolic face; edge and vertex location are inferred from the plane equations. $P'$ in 7(c) results from perturbing the face planes of $F1$ and $F5$, without changing combinatorial structure. $P$ is *simple*, i.e. has no self-intersections, while $P'$ is not.

The winding number of a point $q$ with respect to a polyhedron $Q$ is the sum of the oriented intersections of a ray leaving $q$ with the faces of $Q$. The oriented intersection of the ray with a face $f$ is the winding number of $r$ with respect to $f$, where $r$ is the intersection point of the ray and the face plane of $f$. The endpoint $q$ of the ray is used as a viewpoint, to determine the orientation of cycles on the face plane of $f$. Every interior point of $P$ in in figure 7(a) has winding number $+1$; an interior point of the "tail" of $P'$ has winding number $-1$.

Simplification replaces a possibly self-intersecting polyhedron $Q$ with a set $N(Q)$ of nesting simple shells so that for any point $q$, $w(q, N(Q)) = w(q, Q)$ (A *shell* is a polyhedron with connected boundary.). Simplification first requires that every face be simplified, as described for polygons above. Then for each pair of intersecting faces, the edges induced by intersection with the other face must be determined. This step is identical to face-face intersection required for boolean operations. In figure 7(c), faces $F1$ and $F5$ self-intersect; simplification replaces each by a 4-sided face and a 3-sided face. Faces $F2$ and $F4$ intersect; face-face intersection splits both into a 4-sided face and a 3-sided face. The resulting faces are reassembled into the set $N(P')$ depicted in figure 7(d). The five-faced polyhedron on the left bounds a region of winding number $+1$ and the tetrahedron on the right bounds a region of winding number $-1$. See also figures B.1 and B.2.

As with polygons, the polyhedron bounding the core of $Q$ consists of the shells in $N(Q)$ that separate a region of winding number 0 from a region of winding number $+1$. In figure 7(d), the five-faced polyhedron $P''$ on the left bounds the core of $N(P')$.

**Analysis of rounding.** What can be said about rounding a polyhedron? As is clear from figures 5 and 7, the combinatorial structure of the polyhedron can change arbitrarily (indeed Milenkovic and Nackmann[21] show the NP-hardness of one version of the problem of rounding while preserving structure). Furthermore, a vertex can be ill-conditioned, i.e. its coordinates can change quickly when a defining plane is perturbed.

Nonetheless, it is possible to give a metric bound on the effect of perturbation. Suppose $P$ is an arbitrary (high-precision) simple polyhedron lying within the bounding sphere of radius $2^D$. Then the face planes of $P$ can be rounded to

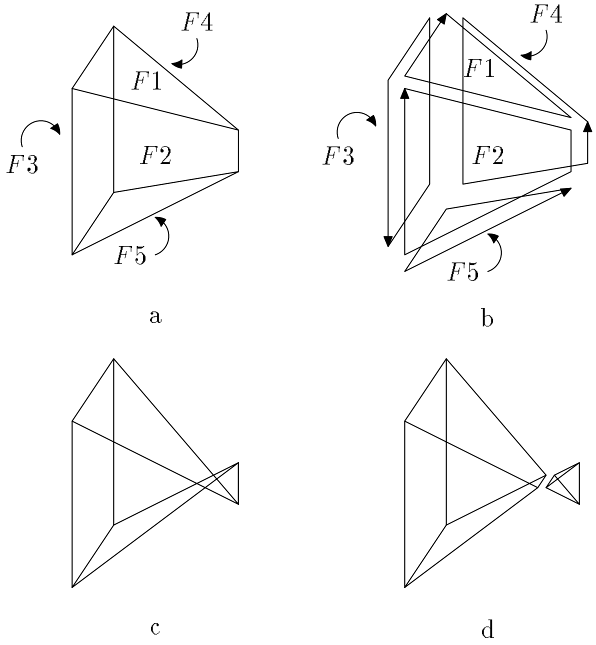



Fig. 7. (a) Wireframe of polyhedron $P$; (b) face cycles; (c) perturbed polyhedron $P'$; (d) set $N(P')$.

bit-length $(B, D)$, so that if $P''$ is the resulting core polyhedron, then any point $p$ in the symmetric difference $P \oplus P''$ lies within a constant times

$$r\chi 2^{-B}$$

of some face of $P$, where $r \geq 1$ upper bounds the radius[1] of any face and $\chi$ is the maximum condition number of any vertex. To achieve this bound, a rounded face plane must closely approximate the original face, in a manner now described. (See the appendix for a proof of the bound.)

### *3.2 Face plane rounding*

Let $\pi$ be the plane of face $f$. A plane $\pi'$ $\delta$*-approximates* $\pi$ at $f$ if the unit normals of $\pi$ and $\pi'$ differ by $\delta$ and if any point of $f$ lies within $\delta(r+1)$ of $\pi'$, where $r$ is the radius of $f$. As long as all points of $f$ are within $2^D$ of the origin, the following strategy finds a plane $\pi'$ of bit-length $(B, D)$ that $\delta$-approximates $\pi$, for $\delta$ a constant times $2^{-B}$.

[1] The *center point* of $f$ is the point of $f$ that minimizes the maximum distance to any other point of $f$; the *radius* of $f$ is the maximum such distance. The *condition number* of a vertex is $1/\phi$, where $\phi$ is the minimum solid angle formed by the three planes defining the vertex.

Choose integers $a$, $b$, $c$ of bit-length $B$ so that $(a, b, c)/\sqrt{a^2+b^2+c^2}$ approximates the unit normal vector of $\pi$. Straightforward rounding guarantees an error bound in the normal of a constant times $2^{-B}$. Choose $d$ so that the center point of $f$ is as close to the plane $\pi' = (a, b, c, d)$ as possible; the separation is a constant times $2^{-B}$ as we can assume one of $|a|$, $|b|$, or $|c|$ is at least $2^{B-1}$. Then the distance of any point of $f$ to $\pi'$ is at most a constant times $2^{-B} + r2^{-B}$.

Lattice basis reduction [18] can often reduce the error in the normal to about $2^{-4B/3}$.

## 4 Implementation

A bare-bones modeler was constructed in C++ using the approach as described. The experiments below were performed on an SGI R3000 running at 40Mhz. The bit-length was (31,22), i.e. 31 bits for each of $a, b, c$ and 53 bits for $d$ in the plane coefficients $(a, b, c, d)$; this yields a 'universe' of diameter about $10^6$. Vertex coordinates, except for the final 'weight' coordinate, had bit-length 120 bits; the weight coordinate had bit-length about 98 bits.

### *4.1 Multiprecision integer arithmetic*

All geometric primitives were implemented using LN [8,9], which provides extended-precision integer arithmetic in a form specially tuned for geometric algorithms. LN is a preprocessor: its input is the specification of an integer polynomial and the bit-lengths of the variables; its output is C++ code that efficiently evaluates the polynomial.

The orientation test and the dot-product evaluation are predicates, where the sign of a polynomial determines control flow. The C++ code generated by LN uses a two-level evaluation strategy. First, the polynomial is evaluated in floating-point arithmetic. If the magnitude of the resulting value is larger than an error bound, the sign of the value is correct and is returned. If not, then the polynomial is evaluated exactly using extended-precision integer arithmetic to determine its sign.

LN assists with both steps. For exact evaluation, LN uses the structure of the expression and bit-length information to generate efficient code. For example, LN estimates the size of intermediate values; this allows temporary storage to be allocated statically, rather than dynamically. Similarly, operations specific to arithmetic, such as carry propagation, can be simplified.

| Primitive | approx | (20,10) | (31,22) |
|---|---|---|---|
| dot product | 7 | 20 | 120 |
| plane orientation | 46 | 230 | 670 |
| point from three planes | 38 | 260 | 670 |

Fig. 8. Floating-point operation counts. First column is standard floating-point; second and third columns are exact evaluation at indicated bit-length (operation counts are approximate).

LN uses double-precision floating-point arithmetic for multiprecision integer arithmetic, since double-precision floating-point arithmetic has longer bit-length and is faster than native integer arithmetic on many current workstations. Figure 8 gives typical operation counts for various primitives, both for approximate and exact evaluation.

The error bound on the initial floating-point evaluation is determined using the structure of the expression, the bit-length bounds on variables, and the error bound on floating-point arithmetic. A two-step error bound gives the best performance. First, a constant error bound is used, determined statically using the worst-case estimates of variable magnitude; the runtime cost of this error bound check is two comparisons per predicate evaluation. If the magnitude of the computed value is less than the constant error bound, a tighter error bound is computed, using the actual magnitude of variables. Typically, the cost of computing this error bound is about the same as the cost of computing the original expression. (LN currently only provides the first bound automatically; the second bound was added manually.)

The two-level evaluation strategy is not used for geometric constructors, such as the computation of the coordinates of a vertex from three planes. The LN-generated code evaluates vertex coordinates exactly. However, a subsequent predicate on the coordinates is evaluated first in floating-point, with exact coordinates rounded to floating-point, and only if necessary is the predicate evaluated exactly.

### *4.2 Experimental results.*

We chose a convex polyhedron with about 250 sides (obtained by intersecting randomly rotated unit cubes). The polyhedron was rotated by an angle in the range $10^{-2}$ radians to $10^{-9}$ radians and then intersected with itself. Dot-products were monitored during the calculation. Figure 9 plots the percentage of dot-products where the floating-point filter could not resolve sign, and hence

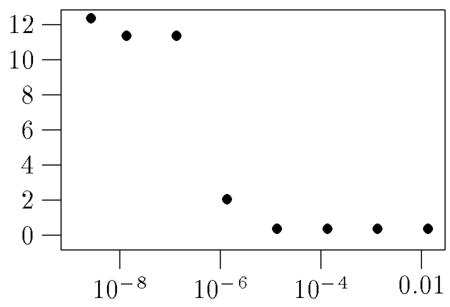



Fig. 9. Percentage required exact dot-products as a function of rotation angle.

exact evaluation was required. The total running time increased from about 8 seconds at a rotation angle of $10^{-2}$ to about 10 seconds at $10^{-9}$. The percentage of running time devoted strictly to arithmetic increased from about 30% to 45% over the same range.

We do not know a natural example where the floating-point filter is significantly less effective. To test hypothetical worst-case behavior, the floating-point filter was removed. Then the intersection takes about 20 seconds, independent of angle. (This time would be larger with a less efficient integer arithmetic package.)

### *4.3 Other work*

Benouamer *et al* [1,16] report on a polyhedral modeler implemented with exact rational arithmetic. Like the modeler described here, their modeler is free of the possibility of numerical error. However, they made different design choices, and it is interesting to compare the results.

Benouamer *et al* use vertex coordinates as the primary geometric representation of a polyhedral solid. This leads to slightly larger growth in the bit-length of coordinate data; they estimate that a computed intersection vertex has about seven times the bit-length of an input vertex. Their implementation uses 'lazy' exact arithmetic, similar in philosophy to the two-level evaluation strategy strategy outlined above. 'Lazy' arithmetic uses floating-point interval arithmetic as a filter, resorting to exact rational arithmetic only when necessary, for example, when a sign-evaluation is required but the floating-point interval contains zero. Their implementation of lazy arithmetic overloads the C++ arithmetic operators. This provides a convenient programming interface but imposes overhead on each arithmetic operation, for the accumulation of error bounds and state-saving [9,16].

An experiment performed by Benouamer *et al* is to intersect randomly-oriented cubes centered at the origin. On an HP/Apollo 33 Mhz 68040, computing the intersection of 20 cubes took about 20 seconds with pure floating-point arithmetic, about 167 seconds using 'lazy' arithmetic, and about 98000 seconds using rational arithmetic with no interval-arithmetic filter. (This is with a modeling accuracy of $10^{-9}$.) The modeler in this paper took about 4 seconds for a similar experiment (on the 40 Mhz SGI); no predicate evaluation required exact arithmetic. Benouamer *et al* do not report timings on nearly-degenerate problem instances. It is plausible that such instances are quite expensive, since they likely require many uses of exact arithmetic.

## 5 Discussion

Serious use of a modeler such as this one would require attention to various engineering issues. For example, typical input data, e.g. from another modeler, is likely to have floating-point coordinates and hence minor numeric inconsistencies. The inconsistencies would be preserved with straightforward translation to integer coordinates. As long as the combinatorial representation is consistent, simplification can be used to obtain a numerically consistent (i.e. simple) polyhedron from a numerically inconsistent (i.e not simple) polyhedron; of course, the combinatorial structure may change.

A polyhedron might not be trihedral; for example, polyhedral models arising in graphics often have triangular faces and hence high-degree vertices. Consider the faces incident to a high-degree vertex. If integer-coordinate face planes are independently obtained from floating-point vertex data, there is no guarantee that all face planes will meet at a common point; even if they are all coincident, symbolic perturbation will remove the coincidence. The simplification algorithm can be used to obtain a simple polyhedron where each high-degree vertex is replaced with a tree of closely spaced degree-three vertices. This replacement is unexpected, and while it causes no problems for the exact-arithmetic modeler, it might for subsequent processing steps.

The rounding algorithm can change combinatorial structure. Usually, the change affects only very small features; for example, rounding may alter the tree of degree-three vertices replacing a high-degree vertex. In consequence, an application cannot assume that all features are preserved by transformations.

### *5.1 Exact arithmetic for other algorithms.*

The use of software exact arithmetic is an easy way to obtain numerically reliable implementations of geometric algorithms. It is appropriate if geometric primitives have small degree and hence minimal bit-length requirements; this covers many algorithms involving linear objects in two and three dimensions.

The exact implementation of geometric predicates on algebraic curves and surfaces appears to require computation on algebraic numbers; it is currently very unclear whether such computation can be made fast enough to be practical. Standard techniques can reduce algebraic number computation to computation with integer polynomials, but estimates of the required arithmetic bit-length can be dauntingly high. Yu [29] analyzes an analogue of the orientation test for conic surfaces implemented using Sturm sequences. He estimates that the intermediate arithmetic bit-length required for the computation is roughly 250,000 times the input bit-length, certainly unthinkable.

It is possible that such bit-length estimates are excessively pessimistic, either because there are better predicate evaluation methods or because instances requiring long bit-length are infrequent. As an example, Burnikel *et al*[2] consider the incircle predicate on points and line segments. Direct application of classical root separation bounds gives a bit-length estimate of about 9000 times input bit-length; a special-purpose argument reduces the bound to 48 times input bit-length; empirically, 6 to 9 times input bit-length appears to be sufficient. Perhaps exact arithmetic can be made to be practical for a restricted class of surfaces, e.g. quadrics, even if not for general algebraic surfaces.

## References


[1] M. O. Benouamer, D. Michelucci, B. Peroche, Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation, *Computer-Aided Design* 26(6):403–416.

[2] C. Burnikel, K. Mehlhorn, S. Schirra, How to compute the Voronoi diagram of line segments: theoretical and experimental results. *Proc. 2nd Eur. Symp. Alg. (ESA 94)*, 1994.

[3] H. Edelsbrunner, E. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, *ACM Trans. Graphics* **9**:66–104, 1990.

[4] S. Fang, B. Bruderlin, X. Zhu, Robustness in solid modeling – a tolerance-based, intuitionistic approach, *CAD*, **25**(9), 1993.

[5] S. Fortune, Progress in computational geometry, in *Directions in Geometric Computing*, Ch. 3, pp. 81–128, R. Martin, ed. Information Geometers Ltd, 1993.

[6] S. Fortune, Numerical stability of algorithms for 2D Delaunay triangulations, *IJCGA*, **5**(1&2):193–213, 1995.

[7] S. Fortune, Polyhedral modeling with exact arithmetic, *Proc. Third Symp. Solid Modeling Appl.* 225–234, 1995.

[8] S. Fortune, C. Van Wyk, Efficient exact arithmetic for computational geometry, *Proc. Ninth Ann. Symp. Comp. Geom*, pp. 163–172, 1993.

[9] S. Fortune, C. Van Wyk, Static analysis yields efficient exact integer arithmetic for computational geometry, submitted.

[10] L.J. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Trans. Graphics* **4**(2):74-123, 1985.

[11] J. Heisserman, R. Woodbury, Unary shape operations, *Geometric and product modeling*, P. Wilson, M. Wozny, M. Pratt, ed., North-Holland, Amsterdam, 1993.

[12] M. Higashi, F. Torihara, N. Takeuchi, T. Sata, T. Saitoh, M. Hosaka, Face-based data structure and its application to robust geometric modeling, *Proc. Third Symp. Solid Modeling Appl.*, 235–246, 1995.

[13] C. Hoffmann, *Geometric and Solid Modeling: an Introduction*, Morgan Kauffmann, 1989.

[14] C. Hoffmann, The problems of accuracy and robustness in geometric computation. *Computer* **22**:31–42 (1989).

[15] C. Hoffmann, J. Hopcroft, M. Karasick, Robust set operations on polyhedral solids, *IEEE Comp. Graph. Appl.* **9**(6):50–59, 1989.

[16] P. Jaillon, Proposition d'une arithmétique rationnelle paresseuse et d'un outil d'aide à la saisie d'objets en synthèse d'images, Thèse, Ecole Nationale Superieure des Mines de Saint-Etienne, 1993.

[17] M. Karasick, D. Lieber, L. Nackman, Efficient Delaunay triangulation using rational arithmetic, *ACM Trans. Graphics* **10**(1):71–91, 1990.

[18] L. Lovasz, *An Algorithmic Theory of Numbers, Graphs, and Convexity*, SIAM, 1986.

[19] M. Mäntyllä, An introduction to solid modeling, Computer Science Press, 1988.

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

[21] V. Milenkovic, L. Nackman, Finding compact coordinate representations for polygons and polyhedra, *IBM J. Res. Dev.* **34**(5):752–768, 1990. A version also appeared in *Proc. Sixth Ann. Symp. Comp. Geom.*, 244–252, 1990.

[22] V. Milenkovic, Practical methods for set operations on polygons using exact arithmetic, *Proc. Canadian Comp. Geom. Conf.*, 1995.

[23] A. Robinson, *Non-standard analysis*, North-Holland, 1966.

[24] R. Seidel, The nature and meaning of perturbations in geometric computing, manuscript, 1993.

[25] J. Stolfi, Oriented projective geometry: a framework for geometric computations. Academic Press, 1991. See also, J. Stolfi, Oriented projective geometry, *Proc. 3rd Ann. Symp. Comp. Geom.*, pp. 76–85, 1987.

[26] K. Sugihara, M. Iri, Construction of the Voronoi diagram for one million generators in single precision arithmetic, First Canadian Conference on Computational Geometry, Montreal, Canada, 1989.

[27] K. Sugihara, M. Iri, A solid modeling system free from topological inconsistency, *J. Inf. Proc., Inf. Proc. Soc. of Japan* **12**(4): 380–393, 1989.

[28] C. Yap, T. Dubé, The exact computation paradigm, 452–492, *Computing in Euclidean geometry*, D.Z. Du, F. Hwang, eds, World Scientific, 1995, second edition.

[29] J. Yu, Exact arithmetic solid modeling, Ph.D. Thesis, Purdue University, 1991. CSD-TR 92-037.

## A Appendix: polyhedra

This technical appendix describes the generalization of polyhedra in more detail. Some proofs are omitted, though no proof is hard.

### *A.1 Nonstandard analysis*

In order to define perturbation cleanly, we use nonstandard analysis [23]. Let $R^*$ be the nonstandard reals. $R^*$ contains *infinitesimals*, whose magnitude is smaller than any positive real, and *infinite numbers*, whose magnitude is larger than any real. For our purposes, it suffices to choose a distinguished infinitesimal $\epsilon > 0$ and use the elements of $R^*$ formed by real polynomials in $\epsilon$. Notice that $\epsilon > \epsilon^2 > \epsilon^3 > \ldots > 0$ (and all are infinitesimals), and that $\epsilon > r\epsilon^2$ for any real $r > 0$.

Concepts of geometry, such as points, lines, and planes, can be interpreted in $(R^*)^3 = R^{*3}$ just as in $R^3$. For example, a plane in $R^{*3}$ is the set of points $\{(x, y, z) \in R^{*3} : ax + by + cz + d = 0\}$, for some $a$, $b$, $c$, $d \in R^*$ with $a, b, c$ not all zero. We view $R^3$ as embedded in $R^{*3}$; a *real* point, plane, etc., has coordinates chosen from $R$.

Let $\pi$ be a real plane and $j$ an integer. A plane $\pi'$ in $R^{*3}$ is an $\epsilon^j$ *perturbation* of $\pi$ if it is a translation of $\pi$ by an amount $r\epsilon^j$ in either direction orthogonal to $\pi$, where $r$ is a real number (in other words, if $\pi$ has real coordinates $(a, b, c, d)$, then $\pi'$ has coordinates $(a, b, c, d + r\epsilon^i)$ for some real $r$.). No real point lies on $\pi'$. For $i = 1, \ldots, 4$ and distinct integers $j_1, \ldots, j_4$, let $\pi'_i$ be an $\epsilon^{j_i}$ perturbation of real plane $\pi_i$. It is easy to see that the four perturbed planes cannot meet at a common point, and that no three perturbed planes meet in a common line.

### *A.2 Combinatorial polyhedra*

A *combinatorial polyhedron* $C = (V, E, F)$ consists of finite sets $V$ of *vertices*, $E$ of *edges*, $F$ of *faces*, together with functions $\mathbf{next}: E \to E$, $\mathbf{sym}: E \to E$, $\mathbf{face}: E \to F$ and $\mathbf{vertex}: E \to V$ satisfying

(i) $\mathbf{next}$ is one-one
(ii) $\mathbf{face}(e) = \mathbf{face}(\mathbf{next}(e))$
(iii) $\mathbf{vertex}(e) = \mathbf{vertex}(\mathbf{sym}(\mathbf{next}(e)))$.
(iv) $\mathbf{sym}(\mathbf{sym}(e)) = e$
(v) $\mathbf{face}(\mathbf{sym}(e)) \neq \mathbf{face}(e)$
(vi) $\mathbf{vertex}(e) \neq \mathbf{vertex}(\mathbf{sym}(e))$.

(For a similar definition, see [10]). $F$ is indexed with distinct positive integers; if we write $f_i \in F$, then $i$ is its index.

Informally an edge is directed, with **face** giving the face to its left, **vertex** the vertex at its head, and **sym** the oppositely directed edge. Edges form cycles under **next** so that each cycle has constant value of **face**; a single such cycle is a *face cycle*. Similarly edges form *vertex cycles* under $\mathbf{next} \circ \mathbf{sym}$.

Face $f$ and edge $e$ are *incident* if $f = \mathbf{face}(e)$ or $f = \mathbf{face}(\mathbf{sym}(e))$; similarly vertex $v$ and edge $e$ are *incident* if $v = \mathbf{vertex}(e)$ or $v = \mathbf{vertex}(\mathbf{sym}(e))$; finally vertex $v$ and face $f$ are *incident* if there is an edge to which they are both incident. Faces $f, f'$ are *adjacent* if there is an edge $e$ with $\mathbf{face}(e) = f$ and $\mathbf{face}(\mathbf{sym}(e)) = f'$. We let $\mathbf{edges}(f)$ denotes the set $\{e : \mathbf{face}(e) = f\}$.

Henceforth we assume that every combinatorial polyhedron is *trihedral*, that is, there are exactly three faces incident to each vertex.

Let $C$ be a combinatorial polyhedron. A map $\Pi$ from faces to planes in $R^{*3}$ is an *embedding* if (1) for each face $f_i$, $\Pi(f_i)$ is an $\epsilon^i$ perturbation of some real plane, and (2) adjacent faces are assigned nonparallel planes. We extend $\Pi$ to vertices and edges: for vertex $v$, $\Pi(v)$ is the point of intersection of the three planes incident to $v$, and for edge $e$, $\Pi(e)$ is the line segment connecting the embedded endpoints of $e$. Clearly $\Pi(\mathbf{edges}(f))$ is a set of closed cycles of line segments, one for each face cycle of $f$, all lying on the common plane $\Pi(f)$. Henceforth we use "edge" or "vertex" to refer also to an embedded edge or vertex. The pair $(C, \Pi)$ is an *embedded combinatorial polyhedron* or simply *polyhedron*.

### *A.3 Winding number*

We write $w(p, c)$ for the (planar) winding number of $p$ with respect to closed curve $c$ (or set of closed curves); the winding number is defined only if $p$ is not on $c$. The definition extends to three dimensions if $p$ and $c$ lie on a common plane; to determine sign, a viewpoint not on the plane must be specified. If $p \in \Pi(f)$, $w(p, f)$ is the winding number of $p$ with respect to $\mathbf{edges}(f)$, as directed by following $\mathbf{next}$.

Let $P = (C, \Pi)$ be a polyhedron and $p$ a point $R^{*3}$. For $r$ a ray in $R^{*3}$ with endpoint $p$, the winding number of $p$ with respect to $P$ using $r$ is

$$\sum_f w(r \cap \Pi(f), f)$$

where the the viewpoint is $p$ and the sum runs over all faces $f$ of $P$ with $r$ intersecting the plane $\Pi(f)$.

**Lemma 1** *The winding number of any real point is defined for any real ray and does not depend upon choice of ray.*

Henceforth we write $w(p, P)$ for the winding number of $p$ with respect to $P$.

### *A.4 Simplicity*

Let $f$ be a face of polyhedron $P = (C, \Pi)$. Face $f$ is *simple* if $\{w(p, f) : p \in \Pi(f)\}$ is $\{0, 1\}$ or $\{0, -1\}$. The *support* of a face is the closure of the set of points of nonzero winding number. A face is *connected* if its support is connected. The intersection of two edges is *proper* if the intersection is at a vertex incident to both edges.

A polyhedron $P$ is *simple* if $\{w(p, f) : p \in R^{*3}\}$ is $\{0, 1\}$ or $\{0, -1\}$. The *support* of a polyhedron is the closure of the set of points of nonzero winding number. A polyhedron is *connected* if its support is connected. The *boundary* of a polyhedron is the union of the supports of its faces; a polyhedron is a *shell* if its boundary is connected. The intersection of two simple faces is *proper* if the intersection of their supports is $\pi(A)$, where $A$ is the set of edges incident to both.

**Lemma 2** *A shell is simple if every face is simple and every intersection between faces is proper.*

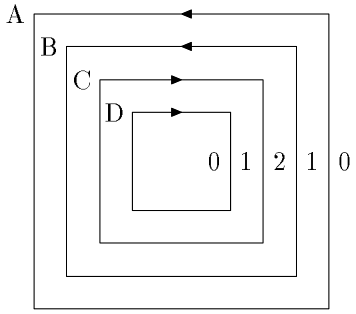



Fig. A.1. Polyhedral nesting (schematic); indices give winding number of any point in the region.

### *A.5 Nesting*

Simple polyhedra $Q$, $Q'$ are *disjoint* if there is no point with nonzero winding number with respect to both. $Q$ *nests* inside $Q'$ if for all $p \in R^{*3}$, $w(p,Q){\neq}0$ implies $w(p,Q'){\neq}0$; $Q$ *properly nests* inside $Q'$ if $w(p,Q){\neq}0$ implies $w(p,Q') = w(p,Q)$. In figure A.1, shells $B$, $C$, and $D$ nest inside $A$, but only $B$ nests properly inside. A set $N$ of simple polyhedra is a *nesting set* if for every pair $Q, Q' \in N$, either $Q$ and $Q'$ are disjoint, $Q$ nests inside $Q'$ or $Q'$ nests inside $Q$; $N$ is a *properly nesting set* if furthermore every nesting pair is a properly nesting pair. We define $w(p,N) = \sum_{Q \in N} w(p,Q)$.

A nesting set $N$ of shells can always be restructured into a properly nesting set $M$ of polyhedra, so that the winding number of any point is preserved. To see this, for integer $i{>}0$, notice that some subset of $N$ bounds $\{p{\in}R^{*3} : w(p,N){\geq}i\}$ (as long as this set is not empty); let $Q_i$ be the polyhedron obtained by merging these shells. Similarly, for $i{<}0$, let $Q_i$ bound the set $\{p{\in}R^{*3} : w(p,N){\leq}i\}$. Then the desired set $M$ is $\{Q_i\}$. In figure A.1, the polyhedron with shells $B$ and $C$ nests properly inside the polyhedron with shells $A$ and $D$. The computation of $M$ from $N$ is straightforward given a data structure that represents the nesting structure of $N$; details are omitted.

## B Simplification

The *simplification problem* for a polyhedron $P$ is to compute a properly nesting set $N(P)$ of simple polyhedra so that for any real point $p$, $w(p,P) = w(p,N(P))$. Clearly, given the set $N(P)$ and an integer $k$, it is easy to extract a simple polyhedron bounding points of winding number exactly $k$, or at least $k$, etc.

Simplification is a slight generalization of usual Boolean operations [11]. For example, if $P$ and $P'$ are simple polyhedra, $P \cup P'$ and $P \cap P'$ can be obtained by merging $P$ and $P'$, simplifying, and extracting points of winding number at least 1, or exactly 2, respectively.

### *B.1 Data structures*

A data structure that represents a combinatorial polyhedron can be obtained directly from the definition: for example, each edge is represented by a node with pointers for **sym**, **next**, **vertex**, and **face** (and probably the inverse of **next**). The node for a face $f_i$ stores coefficients $a_i$, $b_i$, $c_i$, $d_i$, a perturbation direction $\phi_i = \pm 1$, and an index $i$; this represents the plane with coefficients $(a_i, b_i, c_i, d_i + \phi_i \epsilon^i)$.

### *B.2 The simplification algorithm*

The simplification algorithm at a high level is similar to the shell-based algorithm for boolean operations on polyhedra[13].

The simplification algorithm has three steps. First, each nonsimple face $f$ is replaced with a properly nesting set of faces $N(f)$ (defined similarly to a properly nesting set of polyhedra). Each face in $N(f)$ is assigned a distinct index. Hence subsequent steps treat the faces of $N(f)$ as lying on distinct parallel planes. The second step is face-face intersection, which ensures that every intersection between faces is proper. The result of face-face intersection is a nesting set of shells; the final step is to restructure the shells into a properly nesting set of polyhedra.

For an example, consider the self-intersecting box in figure B.1(b); it has the same combinatorial structure as the ordinary box in figure B.1(a). Face simplification splits both the top face and the bottom face into two, and perturbs the pieces with respect to each other (figure B.1(c)). Face-face intersection discovers the improper intersection between the two side faces, and the result is shown in figure B.1(d). No restructuring is needed in this example. Figure B.2 shows another example of simplification.

### *B.3 Face simplification*

A face $f$ is first replaced with a nesting set of simple cycles; to do so it suffices to ensure that the intersection between any pair of edges is proper. Then the cycles are restructured into a properly nesting set of faces $N(f)$; this second step is not discussed further.

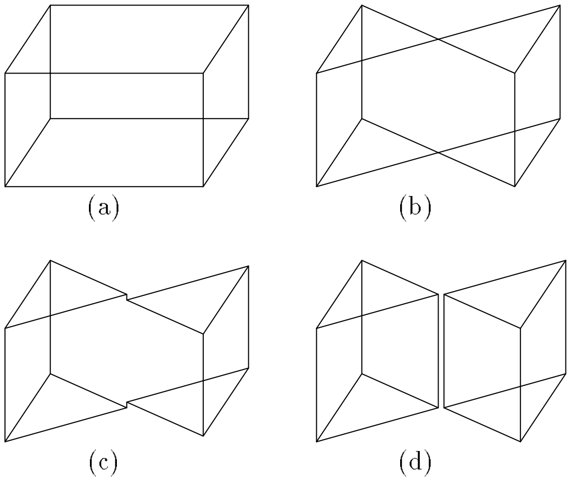



Fig. B.1. (a) box; (b) self-intersecting box; (c) result of face simplification; (d) result of face-face intersection.

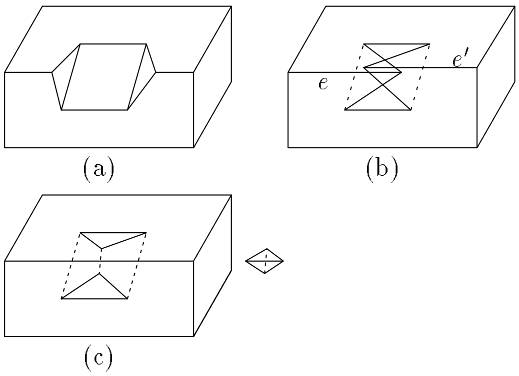



Fig. B.2. The polyhedron in (b) has the same combinatorial structure as the notched polyhedron in (a). Edges $e$ and $e'$ overlap, and both are incident to the same faces. The two polyhedra in (c) are the result of simplification, shown side-by-side for clarity.

The first step is complicated by the possibility of edge overlap. In figure B.2(b), edges $e$ and $e'$ overlap; notice they must be incident to the same face $g$. (If $e$ and $e'$ were incident to distinct faces, then perturbation guarantees that they are nonoverlapping).

First notice that if $e$ and $e'$ overlap but are oppositely directed, then $\mathbf{sym}(e)$ and $\mathbf{sym}(e')$ also overlap, and the local transformation in figure B.3(a)-(b) applied to both faces eliminates the overlap. Hence we can assume that any two overlapping edges are similarly directed.

To eliminate overlap, each edge on face $f$ is assigned a distinct index for the duration of face simplification. Each edge is conceptually perturbed orthogonal to its direction by an infinitesimal amount proportional to its index. The conceptual perturbation is used to determined whether edges intersect. As a

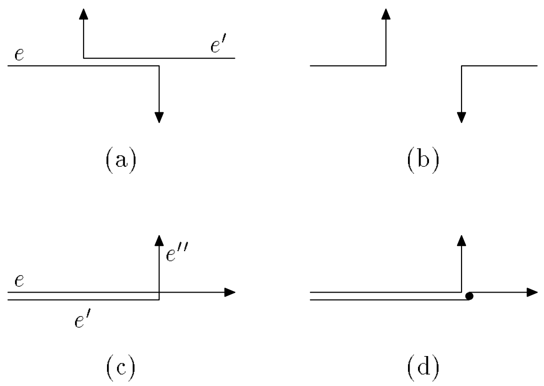



Fig. B.3. Overlapping but oppositely directed edges on the same face (a) can be eliminated easily (b). In (c), overlapping and similarly directed edges have been conceptually perturbed; the intersection is removed as in (d).

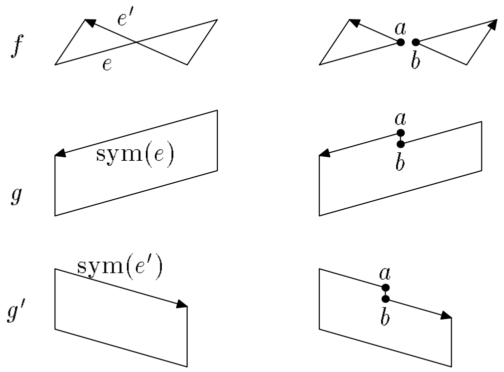



Fig. B.4. Edges $e$ and $e'$ cross on face $f$ (top left). The transformation to eliminate crossing is shown on right, for face $f$ and adjacent faces $g$ and $g'$.

result of perturbation, there are essentially two cases of improper intersection.

The first case is an improper intersection between an edge $e$ incident to a face $g$ and another edge $e''$ adjacent to an edge $e'$ also incident to $g$. See figure B.3(c). Edge $e$ can be split at the intersection point and face $f$ locally transformed as shown in figure B.3(d); the subpieces of a split edge retain the original index. Notice that face $g$ is unchanged (except that one of its edges is split by a degree-two vertex).

The second case is that edges $e$ and $e'$ intersect improperly at a point interior to both edges. Let $g$ and $g'$ be the other faces incident to $e$ and $e'$, respectively. The improper intersection is eliminated by the transformation illustrated in figure B.4. This transformation splits edges $e$, $e'$, $\mathbf{sym}(e)$ and $\mathbf{sym}(e')$ using two new vertices $a$ and $b$. Vertices $a$ and $b$ will end up on different faces in $N(f)$; to guarantee that the result of face simplification is a combinatorially valid polyhedron, it is necessary to add an edge between $a$ and $b$ on both faces $g$ and $g'$.

### *B.4 Reindexing*

Suppose $e$ is an edge on face $g$ also incident to face $f$. As a consequence of simplifying $f$, $e$ may have been replaced with a sequence of edges $e_0$, $e_1$, …, $e_{2k}$, where the even-numbered edges are now incident to faces in $N(f)$ (see figure B.4). We must ensure that this replacement introduces no new improper edge intersections on face $g$, since $g$ may already have been simplified. This follows if the indices assigned to faces in $N(f)$ have the same order relation with other face indices as $f$ does. This can be accomplished easily with a global reindexing of all faces, though other strategies are possible.

Because of the change in indices, the winding number of nonreal points need not be preserved by simplification.

### *B.5 Face-face intersection*

Face-face intersection ensures that all intersections between faces are proper. The implementation of this step is reasonably standard [13] (it would not be standard if faces were not simple). A detail is that even if faces $f$ and $g$ are already adjacent, face-face intersection is necessary to discover all possible common edges.

## C The rounding bound

The bound on polyhedral rounding claimed earlier is a consequence of the following lemma.

**Lemma 3** *Let $P{=}(C,\Pi)$ and $P'{=}(C,\Pi')$ be polyhedra, where for each face $f$, $\Pi'(f)$ is a $\delta$-approximation of $\Pi(f)$ at $f$. Then for any point $p{\in}R^{*3}$, if $w(p,P){\neq}w(p,P')$, then $p$ lies within $\kappa\delta\chi R$ of some point of the boundary of $P$, where $\kappa$ is a constant, $\chi{<}1/(2\delta)$ is the maximum condition number of any vertex of $P$, and $R{\geq}1$ bounds the radius of any face of $P$.*

**Proof sketch:** We decompose $P'$ into a set $N$ of polyhedra, one of which is $P$. For simplicity, assume that all face planes of $P$ and $P'$ are distinct and none are parallel.

Let $Q$ be a polyhedron with face $f$ lying on plane $\lambda$, and let $Q'$ be the polyhedron obtained by embedding $f$ on plane $\lambda'$ instead. The *difference polyhedron* $D$ obtained by offsetting $\lambda$ to $\lambda'$ satisfies $w(q,D) + w(q,Q) = w(q,Q')$ for all $q \in R^{*3}$; explicitly $D$ has two copies of face $f$, one embedded on plane $\lambda$, the

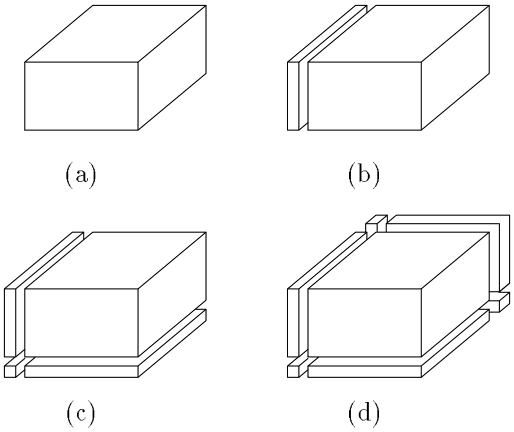



Fig. C.1. (a) a polyhedron $Q$; difference polyhedra obtained by offsetting left face plane (b), bottom face plane (c), and back face plane (d).

other oppositely directed and embedded on plane $\lambda'$, and for each edge $e$ of face $f$, a four-sided face embedded on the plane of the other face incident to $e$. See figure C.1(a)-(b).

The desired set $N$ of polyhedra is obtained by starting with $P$, and for each face $f$ of $P$ in turn, adding to the set the difference polyhedra obtained by offsetting $\Pi(f)$ to $\Pi'(f)$. See figure C.1(c)-(d). A polyhedron in $N$ is either $P$ itself, a *face offset* with two copies of a face of $P$, an *edge offset* with four copies of an edge of $P$, or a *vertex offset* with eight copies of a vertex of $P$.

To prove the lemma, suppose $w(p, P) \neq w(p, P')$. Since we have $w(p, P') = w(p, N)$, there must be some polyhedron $Q$ distinct from $P$ in $N$ with $w(p, Q) \neq 0$. We assume $Q$ is a face offset of some face $f$; other cases are similar.

$Q$ has a face $g$ that is a copy of $f$ lying in $\Pi'(f)$, and a face $g'$ with support identical to the support of $f$. Using some elementary geometry and the definition of condition number, it is possible to show that any vertex of $g$ lies within $\kappa_1 R\delta\chi$ of the corresponding vertex of $g'$ and within $\kappa_2 R\delta$ of $\Pi(f)$, for some constants $\kappa_1, \kappa_2$. Let $p'$ be the orthogonal projection of $p$ onto plane $\Pi(f)$; segment $pp'$ has length at most $\kappa_2 R\delta$. Either segment $pp'$ meets no face of $Q$, and $p'$ lies in the support of face $g$, or segment $pp'$ meets a four-sided face of $Q$, and there is a path of length at most $\kappa_1 R\delta\chi$ along the four-sided face to an edge of $g'$ . Hence $p$ lies within $\kappa R\delta\chi$ of the support of $g'$, for some constant $\kappa$. $\square$