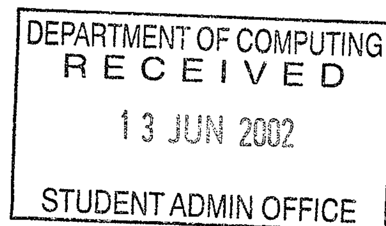
Supervisor = Abbas Edalat.

Robust Polyhedral Rounding

by

Mohammad Raza
(JMC 3)



Abstract

Computational geometry is an area of computer science in which symbolic information is highly sensitive to changes in numerical data. It is therefore a non-trivial problem in solid modeling to develop an efficient and robust algorithm for polyhedral rounding. Based on the domain theoretic framework for solid modeling defined in [1], this project presents such an algorithm for rounding a polyhedron defined in terms of its face planes, and gives an analysis of its error and complexity.

Contents

1. Introduction

2. The Need for Rounding
   - 2.1 Problems With Floating Point Arithmetic
   - 2.2 Exact Arithmetic and the need for rounding

3. Geometry of Planes in Three Dimensions
   - 3.1 Determining a plane by its normal vector and distance from the origin
   - 3.2 A uniform set of planes

4. The Rounding Algorithm
   - 4.1 Approximating the normal vector
     - 4.1.1 The 'Fast' method
     - 4.1.1 The 'Accurate' method
   - 4.2 Shifting

5. Error and Complexity Issues
   - 5.1 Bounding the Error
     - 5.1.1 Bounding the Error in the Normal
     - 5.1.2 Distance of the Face from the Rounded Plane
   - 5.2 Complexity

6. Erroneous Situations
   - 6.1 The CSG Approach
   - 6.2 Simplification
   - 6.3 Ensuring Containment

7. Conclusion

8. Acknowledgements

9. Bibliography

1. Introduction

Solid modeling is about the abstraction of the symbolic (topological) and numerical (geometrical) information which describes a polyhedron, as well as the implementation of operations which manipulate both kinds of information, such as affine transformations or boolean operations like union, intersection, etc. Theoretically, these operations are defined over the real space, and so we need a substitution for real arithmetic when implementing them in practice. Simply using floating-point arithmetic can do this, but this approach seriously affects the soundness and consistency of the results of geometrical algorithms in some cases, as we will see later.

We therefore need our computations to give exact results, in order to maintain the validity of the output. This can be done using software extended with exact integer arithmetic. There is, however, the disadvantage that allowing exact results would mean an unbounded growth in the bit-length of our numerical data. This leads us to the central issue of this project: to devise a robust and efficient algorithm that could be used to round the numerical data of a given polyhedron to a desired bit-length, and which gives a geometrically valid result.

As mentioned before, this algorithm would be based on the domain theoretic framework defined in [1]. This framework is designed to overcome another problem in solid modeling, which is the discontinuity of the membership predicate on the boundary of the objects that are modeled. Operations based on the membership predicate are not continuous and therefore not computable. The solution that is proposed in [1] is to redefine the membership predicate so that it maps to the set $\{tt, ff\}_{\perp}$, where the symbol $\perp$ is the result of applying the membership predicate to points on the boundary of the object. Rounding within this framework is defined by two operations, R and R*, which round a solid S to the 'inside' and the 'outside' respectively, i.e. such that S lies in the interval [R(S), R*(S)], and R(S) and R*(S) would have numerical data that would be integers or rationals of a desired bit-length. This is similar to bounding a real number in an interval defined by two rational numbers.

One issue that presents itself at this point is the choice of representation of the polyhedron, as this determines the geometrical attributes of the object that would be the subject of the rounding. Nikolopoulos in [2] describes the implementation of boolean operations and presents a rounding algorithm (based on the above framework) for a polyhedron defined by its vertex coordinates. He also introduces the rounding algorithm that is the subject of this project, where the polyhedron is defined by its face planes. A number of reasons are discussed for preferring this representation to the former.

A plane can be defined by its equation in three dimensions using four numbers ($Ax + By + Cz + D = 0$), while representing the same plane using three of its vertices would require storing nine numbers. If a face polygon consists of more than 3 vertices, then these may not lie on the same plane after the vertex rounding. So the faces would have to be triangulated before the rounding, and then recombined after the rounding if they are still co-planar. These steps are not required in a plane rounding algorithm. Another important advantage of the planar representation is that boolean operations are much faster and simpler to implement, as they mainly work on symbolic information. They would also not produce any new numerical data, and so rounding would not need to be performed subsequently.

Before describing the plane rounding algorithm, it is important that we first discuss the need for and nature of polyhedral rounding. Following that, there is a brief discussion of the geometrical attributes of planes in three dimensions, in order to understand the problem in a geometrical context. We then describe the algorithm in two forms, one, which is aimed at optimising speed, and another in which emphasis is placed on accuracy. The following section looks at error and complexity issues, including a comparison of the two forms of the algorithm. We conclude the report with a discussion of alternative approaches in which the algorithm can be used, and cases in which further operations may be required after applying the algorithm to ensure the validity of the result.

2. The Need For Rounding

This section describes the emergence of what is usually referred to as the 'Exact Computation Paradigm' in computational geometry, and how the requirement of a process of rounding follows as a direct result of this approach. The approach stems from the inadequacy of floating-point arithmetic when used with geometrical algorithms.

2.1 Problems With Floating Point Arithmetic

The adverse effects of using floating point arithmetic for geometrical algorithms are discussed in detail in [2] and [3]. Numerical errors may occur as a result of decimal to binary conversion, rounding the results of successive arithmetic operations, or digit cancellations when finding the difference of two nearly equal numbers. These numerical errors would subsequently affect the geometrical results.

One problem that may be encountered is that of incidence asymmetry. This happens when incidence tests yield inconsistent results. For example, we consider an algorithm that tests the equality of two points, each defined as the intersection of two lines. When given the input (L1, L2) and (L3, L4), it first calculates the intersection of L1 and L2, substitutes the result into the equations of L3 and L4, and returns true if the results are within a

certain tolerance. We find that in some cases the algorithm would give contradictory outcomes if we swap (L3, L4) and (L1, L2), which is clearly illogical (refer to [3] for a concrete example).

Another problem is that of incidence intransitivity. Programmers usually use e heuristics to allow a certain error due to inaccuracy of the numerical results. For example, incidence tests on points may allow two points to be considered equal if they are within a distance of e of each other. Clearly, this definition of the equality predicate would not be transitive, for if we consider three collinear points u, v and w situated at intervals of length e, then we have equal(u,v), equal(v, w), but not equal(u, w).

Topological validity is another cause of concern when using floating-point arithmetic. For example, we consider the case where two edges e1 and e2 are to be tested for intersection. Let e2 be the common edge of faces f1 and f2. The algorithm may proceed by testing the intersection of e1 with either f1or f2. If the angle that e1 makes with f1 is very shallow, it may be deduced that the two edges do intersect. But if the test is done using f2 and the angle between e1 and f2 is not so small, then we may find that the e1 in fact does not intersect e2 and the test would return false. Figure 2.1.1 illustrates the problem.

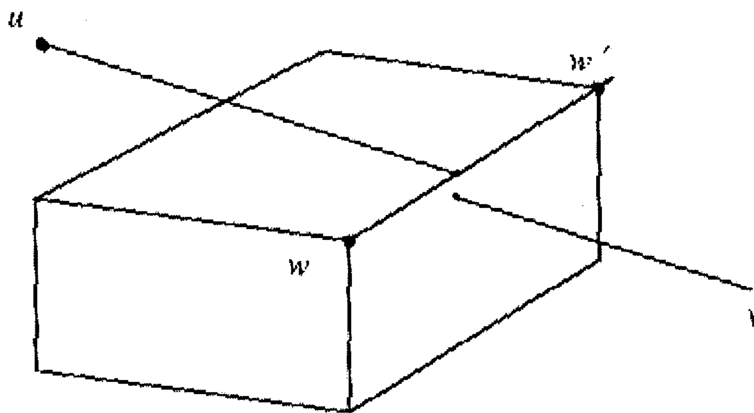


Figure 2.1.1

2.2 Exact Arithmetic and the need for rounding

We have seen that using floating-point arithmetic has significant drawbacks. None of these problems would occur if an exact arithmetic is used for computation. Geometrical programs therefore usually make use of software implemented exact integer arithmetic, which allows the use of unbounded integers. In this project, we use the GMP multiprecision package ([4]), which is claimed by its developers to be "as fast as possible". Introducing an exact arithmetic, however, would naturally raise issues of complexity. Apart from the inefficiency due to software-implemented precision, we have the problem of unbounded growth of the bit-length of our numerical data as the algorithm proceeds with successive geometrical operations. Citing an example from [2], if the equation of a plane is to be computed using three vertices that lie on it, then the bit-length of the plane coefficients would be nine times that of the coordinates of the vertices. An unbounded numerical data set would also make the output of the programs less compatible with the "outside floating-point world" ([2]). This implies the need for an algorithm that would round the numerical values to a desired bit-length, which would usually be the local hardware bit-length. This is clearly not just a matter of directly rounding the values, because we need to maintain the validity of the combinatorial incidence information of the solid.

It is interesting to reflect on the global issue at this point. It seems that the problem is ultimately not that much different from other areas of computing where rounding may simply be done using floating-point arithmetic, as it is still the case that rounding *is* eventually done, though the process may be more involved in this instance. It is just that in this case, the numerical data is not an independent set of information, which can be manipulated separately, and so the scope of our rounding process is extended to include the symbolic, as well as the numerical information at hand. The use of exact arithmetic can therefore be viewed as an intermediate stage in our 'high level' rounding algorithm.

3. Geometry Of Planes In Three Dimensions

This is the Cartesian equation of a plane in three dimensions is given as

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

In our planar representation of a polyhedron, each face plane would be represented by the four numbers (integers) a, b, c and d. The result of the rounding operation R (or R*) would give a polyhedron whose face planes are such that it would be contained in (or would contain) the original polyhedron. This requires us to understand the meaning of a, b c and d in a geometrical context.

3.1 Determining a plane by its normal vector and distance from the Origin

We note two simple properties of a plane in three dimensions. The first observation is that the vector (a, b, c) is orthogonal (normal) to the plane $\{(x, y, z) : ax + by + cz + d = 0\}$. This can be easily seen by checking that the dot product of any vector on the plane with (a, b, c) is always zero. We now have to understand the meaning of d in a geometrical context.

Proposition *If (a, b, c) is a unit direction vector (the unit normal of the plane) then |d| is the distance of the plane from the origin.*

Proof: The distance of the plane from the origin is the length of the perpendicular from the plane to the origin. Let this be represented by the position vector p = (p_1, p_2, p_3) from the origin to a point on the plane.

As the point p lies on the plane, we have

$$a p_1 + b p_2 + c p_3 + d = 0 \qquad (1)$$

Now, because the vector is perpendicular to the plane, we have

$$p = k(a, b, c) \qquad (2)$$

where |k| is the magnitude of p because (a, b, c) is a unit vector in the direction of p. Substituting this into (1) we get

$$k(a^2 + b^2 + c^2) + d = 0$$

=> $k + d = 0$ as (a, b, c) has magnitude 1

=> $|d| = |k|$

QED.

The last two propositions show that we can identify a plane in terms of its normal vector and distance from the origin, given the Cartesian coefficients a, b, c and d. This information would be very useful in the design of a geometrical rounding algorithm.

3.2 A uniform set of planes

One other factor to consider is the choice of the relative bit-lengths (ranges) of the four coefficients. We would like to map to a set of planes, which are uniformly distributed, as described in [3]. A uniform configuration would be to be

$$-2^n \leq a,b,c \leq 2^n$$

$$-2^{2n} \leq d \leq 2^{2n}$$

so that a, b, and c are represented in n bits, and d in 2n bits. To see the situation in two dimensions (for lines), Figure 3.2.1 and 3.2.2 illustrate an example of the distribution of lines when the bit-length is the same for all coefficients and when the configuration described above is used. It can be seen that the distribution is more uniform in the latter. In homogenous coordinates, the bounds would become uniform, due to the introduction of the 'weight' coordinate.

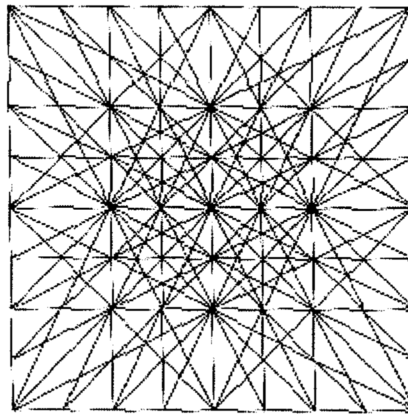


Figure 3.2.1

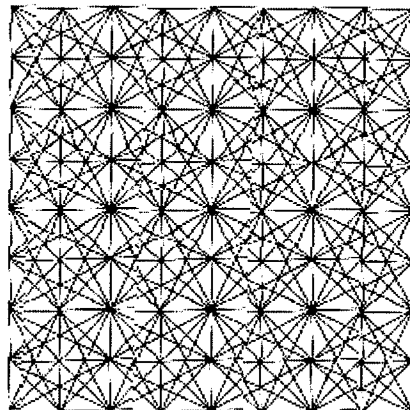


Figure 3.2.2

4. The Rounding Algorithm

The algorithm described here is similar to that proposed by Steven Fortune in his 1996 paper on polyhedral rounding using multi-precision integer arithmetic ([6]). The face plane coefficients are stored as multi-precision integers, but in this case we use the more up to date GMP package, instead of the LN package that is used in [6]. As we are dealing with integer coefficients, for the plane {(x, y, z): ax + by + cz + d = 0}, the size of the integers represents the 'geometrical' precision. For example, a large bound on a, b and c would make for a wider choice of angles for the normal vector. Indeed, using one binary bit would mean that the planes are restricted to have normals in the direction of (up to sign) (0,0,1), (0,1,0), (1,0,0), (1,1,0), (1,0,1), (0,1,1), or (1,1,1). Also, it should be noted that using a bound of 2n bits for d and n bits for a, b and c (as explained in the previous section) requires that the original plane be within a distance of 2^n of the origin. Beyond this limit, the error in d would increase with distance from the origin.

The topological information of the polyhedron can be stored in terms of 'face cycles' as done in [6]. A polyhedron data structure can consist of a list of faces. Each face object contains an index (a natural number used to identify the face), a list of face indices (face cycle) representing the faces that surround this face, and a face plane, which holds the plane coefficients a, b, c and d.

Given such a polyhedron, there are two main steps in the rounding process for each face. Firstly, the coefficients are rounded by approximating the normal vector. This is a local operation on the plane, i.e., independent of the symbolic information of the polyhedron. The next step is to increment (or decrement) the value of d until all the vertices of the original face lie in the positive (or negative) halfspace of the rounded plane, so that the rounded face is 'inside' or 'outside' the original face, depending on whether the operation is R or R*. We would now look at these two steps in more detail.

4.1 Approximating the normal vector

4.1.1 The 'Fast' method

Given the plane {(x, y, z) : ax + by + cz + d = 0}, with normal n = (a, b, c), we want to find n' = (a', b' ,c'), where

$$-2^n < a', b', c' < 2^n$$

that would approximate the direction given by n. This can simply be done in the following way.

Let $m = \sqrt{a^2 + b^2 + c^2}$

$$a' = \left\lfloor (2^n - 1) \times \left(\frac{a}{m}\right) \right\rfloor \quad b' = \left\lfloor (2^n - 1) \times \left(\frac{b}{m}\right) \right\rfloor \quad c' = \left\lfloor (2^n - 1) \times \left(\frac{c}{m}\right) \right\rfloor \quad d' = \left\lfloor (2^n - 1) \times \left(\frac{d}{m}\right) \right\rfloor$$

So the normalizing factor, m, preserves the direction, and then multiplying by $2^n - 1$ minimizes the error due to the truncation that follows. The same operation is applied to d to reduce it down to within its range of 2n bits (for now). Note that if we did not need to bound d within its range of 2n bits, then we could have set m to max(a, b, c) to optimize our use of the grid. Figure 4.1.1 illustrates the possibilities for n' (in two dimensions, for the first quadrant, with n = 2).

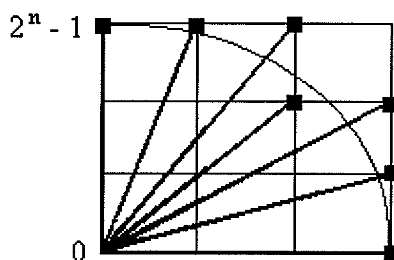


Figure 4.1.1

4.1.2 The 'Accurate' method

It is observed that in the previous method, the available grid of integers may not be fully utilized to give an optimized result. Only the points near to the circumference of the arc in figure 4.1.1 are being used, and all points in the area enclosed by a cube of length $(2^n - 1)/\sqrt{2}$ inside the arc would be ignored. For example, referring to figure 4.2.1, it may be the case that B is at a smaller angle to n than either A or C.

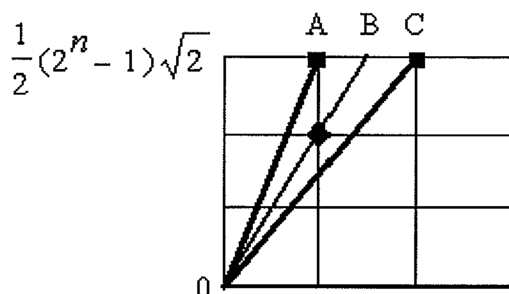


Figure 4.2.1

If the grid is to be fully utilised to account for cases like these, then we can proceed in the following way.

Let $m = \sqrt{a^2 + b^2 + c^2}$ as before.

Set angle to 90 (assuming that the error in the angle does not exceed this, which is the case for n >= 1).

for count from 1 to 2^n -1

$$a' = \left\lfloor (2^n - 1) \times \left(\frac{a}{m}\right) \right\rfloor \quad b' = \left\lfloor (2^n - 1) \times \left(\frac{b}{m}\right) \right\rfloor \quad c' = \left\lfloor (2^n - 1) \times \left(\frac{c}{m}\right) \right\rfloor \quad d' = \left\lfloor (2^n - 1) \times \left(\frac{d}{m}\right) \right\rfloor$$

if the angle between (a, b, c) and (a', b', c') is less then angle, then update angle to this

end for

This ensures that the error in the angle can only be less than or equal to the error when using the previous method. The two methods basically represent two extreme situations, on a range of possible approaches depending on how much of the grid the user wishes to utilize.

4.2 Shifting

Once the normal has been rounded, we have obtained the values of a', b' and c', and d' is within its required range of 2n bits. The plane now has to be 'shifted' so that all the vertices of the original face lie in the positive or negative halfspace, depending on whether the rounding operation is R or R*.

In this planar representation of a polyhedron, all the vertices are given symbolically, i.e. as the intersection of three planes, say p, q and r. Fortune in [6] discusses the orientation test on planes. Given the four planes p, q, r and s, it is required to find the orientation of the vertex v, defined by p, q and r, with respect to the plane s. Consider the matrix M given by

$$\begin{pmatrix} a_p & b_p & c_p & d_p \\ a_q & b_q & c_q & d_q \\ a_r & b_r & c_r & d_r \\ a_s & b_s & c_s & d_s \end{pmatrix}$$

In homogenous coordinates the point v is given by

$$v = (-M1, M2, -M3, M4)\text{sign}(M4)$$

where sign(M4) is 1, -1 or 0 and Mi is the determinant of the 3 by 3 matrix obtained by removing the last row and the ith column from M. We multiply by sign(M4) because we want the weight coordinate of v to be positive (a point in Cartesian coordinates corresponds to two points in homogenous coordinates, one where the weight is positive and the other where it is negative; refer to Stolfi [7]). This implies that the sign of the dot product of s with p would determine whether v lies on the plane s, in its positive halfspace or in its negative halfspace.

Returning to the rounding algorithm, to implement the shifting process we proceed in an iterative manner, incrementing (or decrementing) d, and testing the orientation of all the

face vertices with the rounded plane at each step. The algorithm stops when the orientation of all the vertices is as required (i.e. +1, -1 or 0). This gives us the value of d'.

5. Error and Complexity Issues

The analysis in this section gives us a bound on the error due to the rounding operation(s), and an idea of the complexity of the algorithm in both its forms.

5.1 Bounding the Error

There are two factors to consider when analyzing the error in the result. We will first examine the error bound on the normal vector (i.e., the angle between the two planes), and then calculate a bound on the distance of the face from the rounded plane.

5.1.1 Bounding the Error in the Normal

As described in the last section, the error associated with the second method would be less than or equal to that in the first. So we just need to examine the first method. As

discussed before, the points representing the rounded normals would lie outside a cube of length $(2^n - 1)/\sqrt{2}$ that lies inside the arc. Referring to figure 5.1.1, if we consider the angles $\theta_1 \ldots \theta_3$, then it is observed that θ_1 would be the largest angle among the three. The case is identical for the other three angles (along the right side of the grid). In general, we see that the situation is the same for any value of n greater than two: that the largest possible angle between n and n' cannot exceed θ_1 (in two dimensions).

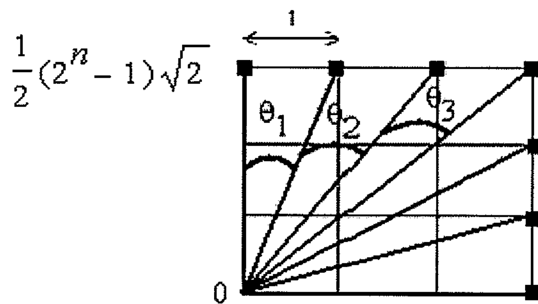


Figure 5.1.1

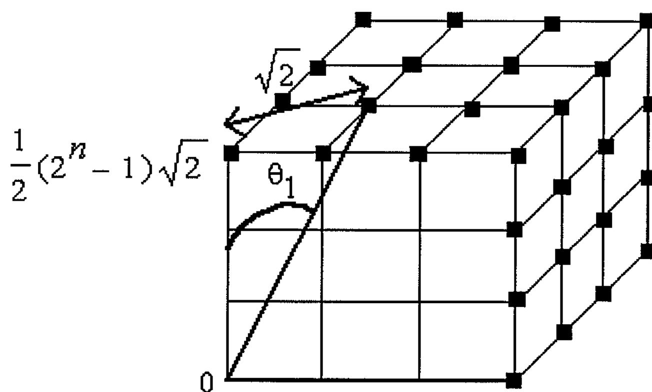


Figure 5.1.2

The three dimensional case is similar, as illustrated in figure 5.1.2. The bound on the angle is therefore calculated as follows:

$$\theta_1 = \arctan\left(2\frac{1}{2^n - 1}\right)$$

5.1.2 Distance of the Face from the Rounded Plane

Having found a bound on the normal, we now proceed to discuss the distance of the original face from the rounded plane once the algorithm has been applied. As described before, the d coefficient of the plane equation is the distance of the plane from the origin when (a, b, c) is the normal vector. In the iterative step of the algorithm, d is incremented by one unit at every iteration until the plane is 'inside' or 'outside' the face. Therefore, in the worst case, the magnitude of the rounded normal vector would be minimal, so that each unit of d represents the largest possible measure of distance, maximizing the distance t as illustrated in figure 5.1.3.

Using the first method, the minimal value of the magnitude of the rounded normal vector would be $2^n - 1$ (it will always be this value because of the way in which the method works), and so t is at most $1 / (2^n - 1)$. The second method requires the adjustment that once the normal has been acquired, we 'expand' the coefficients to their maximum representation within the grid by multiplying them by ($2^n - 1$ div magnitude(n')). In the worst case, this multiplying factor would be one, which means that magnitude(n') would be greater than $2^{n-1} - 1$. So the minimum magnitude of n' is $2^{n-1} - 1$, and t is at most $1 / (2^{n-1} - 1)$.

We immediately see another drawback of using the second method: the bound on t is about twice as much in this case as that when using the first method. So we may have to compensate a closer gradient with a larger distance, and this would not sound very appealing in most applications.

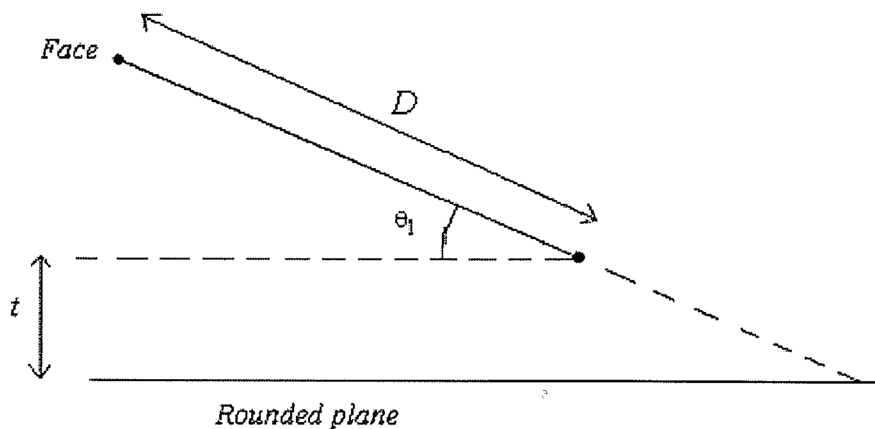


Figure 5.1.3

We define the diameter, D, of a face of a polyhedron as the diameter of the smallest circle that encircles the face. The largest possible angle between the face and the rounded plane is the same as the error bound in the normals, given in the previous section. Therefore, as shown in figure 5.1.3, the maximum distance of any point on the original face to the rounded plane is given by

$$t + D\sin(\theta_1)$$

This method is based on that described by Fortune in [6].

5.2 Complexity

The complexity of the algorithm depends on two factors: the required bit-length to round to (n), and the number of faces of the polyhedron (m). This is of course not taking into account the speed of the multi-precision package that is being used, which is naturally expected to become slower as the size of the integers increases (refer to the GMP manual from [4] for a detailed description of the algorithms and their complexity).

We first consider the number of operations needed for rounding each face. For rounding the normal, using the fast method only involves a constant number of operations, independent of n (O(1)). The second method on the other hand makes for an exponential increase in the number of operations involved in this step ($O(2^n)$).

The next step is the d incrementation (shifting). In the worst case, the number of iterations here is maximized. The angle is at a maximum (as in 5.1.1) and the magnitude of the rounded normal vector is maximized so that each unit of the d coefficient represents the smallest possible shift of the plane. This maximum magnitude is $2^n - 1$ using both methods. So for n bits, this minimum size of each unit of d is

$$\frac{1}{2^n - 1} \qquad (*)$$

Figure 5.2.1 illustrates the shifting process. The difference between the distances from the origin of the rounded and original planes would be at most 2-n (this makes a difference of about one iteration), so we can assume that the circle has radius d as shown. Now the number of iterations largely depends on how far the face is located from the perpendicular of its plane to the origin. As discussed before, it is assumed that the planes are within a distance of 2n –1 from the origin, so we can bound the distance of the face from the

perpendicular by $2^n - 1$. The diagram illustrates the calculation of the perpendicular distance the rounded plane has to be shifted.
This comes out to be approximately

$$(2^n - 1)\sin(\theta_1) + d(1 - \cos(\theta_1)) \qquad (**)$$

.

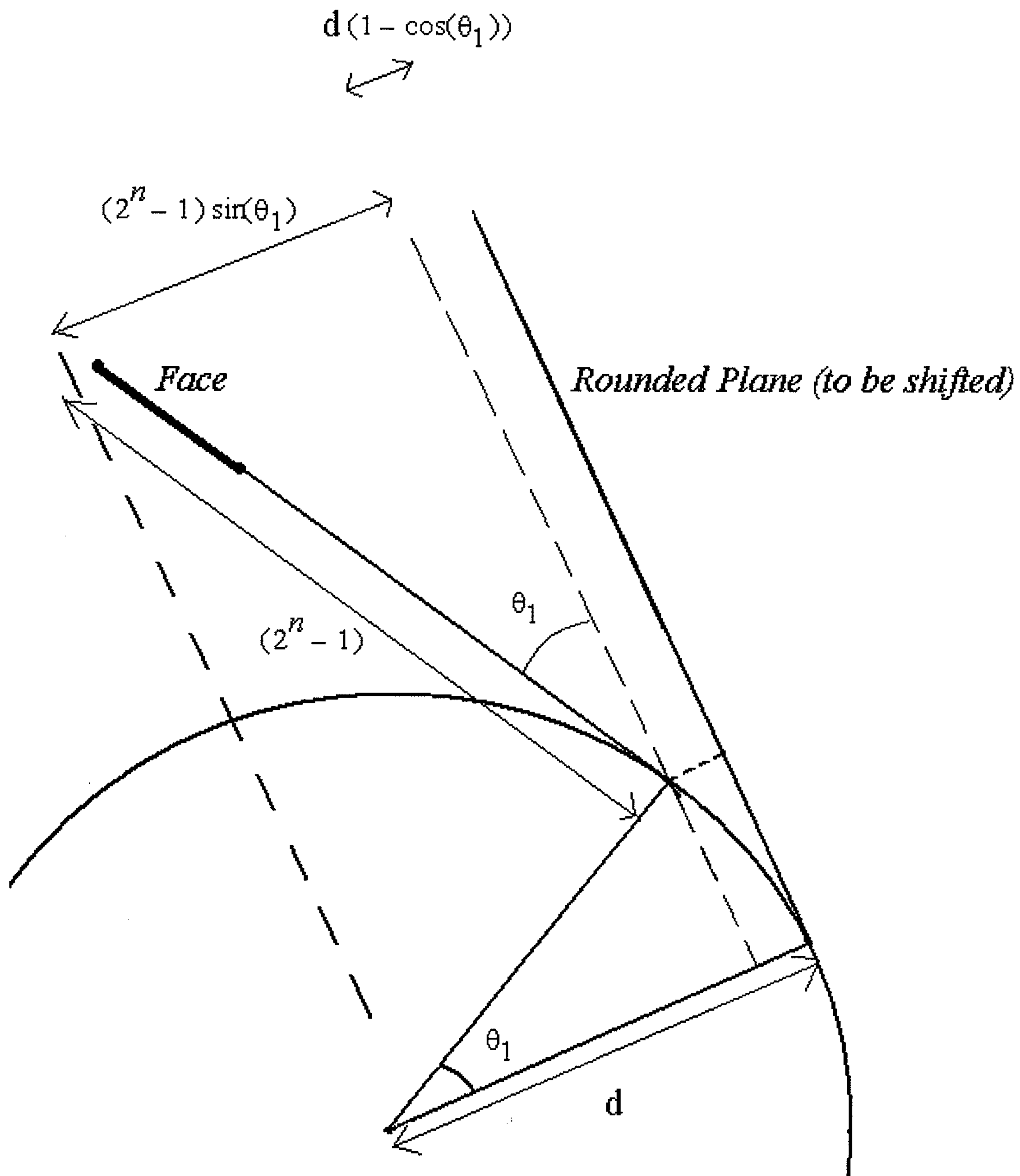


Figure 5.2.1

θ_1 is assumed to be small, so that $\sin\theta_1 \approx \theta_1$, $\cos\theta_1 \approx 1$ and $\tan\theta_1 \approx \theta_1$. Using (*) and substituting the value θ_1 (from 5.1.1) in (**), in the worst case, the number of iterations is approximately given by $2(2^n-1)$.

Finally, we consider the polyhedron as a whole. The algorithm rounds each of the face planes individually, and so the total number of operations is given by

m*(number of iterations needed to round each face)

$= m(k + 2h(2^n-1))$ (for the first method)

and $= m(k.\ 2^n + 2h(2^n-1))$ (for the second method)

where k and h are constants. Note that with the 'shifting' stage included, the two methods have the same average complexity ($O(m*2^n)$).

6. Erroneous Situations

As for most polyhedral rounding algorithms, erroneous cases occur as a result of applying the algorithm when the rounded polyhedron turns out to be non-simple (self-intersecting). Also, as we are working within the framework of [1], we need to ensure containment of the result within the original polyhedron in the case of R, and vice versa in the case of R*. In this chapter, we consider two possible approaches to overcome this problem.

6.1 The CSG Approach

This is the methodology proposed by Sugihara and Iri in [8]. As described in [6], in this approach, every polyhedron is defined as a sequence of Constructive Solid Geometry (CSG) operations on primitive solids. The primitives should be 'well-conditioned' so that changes in the geometry due to the rounding perturbation should not invalidate the topological information. We would discuss in the next section how topological validity is also expected to resolve the containment issues mentioned above.

The way to proceed with the rounding would then be to apply the algorithm on each of the primitives used in the definition of the solid, and then to reapply the CSG operations. As CSG operations are always valid, we can be sure that the rounded polyhedron would be a valid one.

6.2 Simplification

An alternative approach is to use 'simplification', as proposed by Fortune in [6]. A general polyhedron can be defined directly by its face planes and topology, rather than in terms of CSG operations. The definition of a polyhedron is extended to allow the polyhedron to self-intersect. This means that the polyhedron may have 'holes', 'holes within holes' or 'solid sections within solid sections'. This information is represented by the 'winding number' of a point as shown in figure 6.2.1 (for a polygon). In the case of polygonal or polyhedral structures, we don not need the formal definition of the winding number. The winding number of a point with respect to a polygon is calculated by considering a (any) ray from this point to infinity. The winding number is initialised to zero. Every time the ray intersects an edge of the polygon from the 'right', we add 1 to the winding number, and if it is from the 'left' then we subtract one. In the case of a polyhedron, again, a ray is drawn from the point to infinity and the winding number is initialised to zero. At every intersection of the ray with a face of the polyhedron, the winding number of the point of intersection with respect to the face is added to the winding number being calculated.

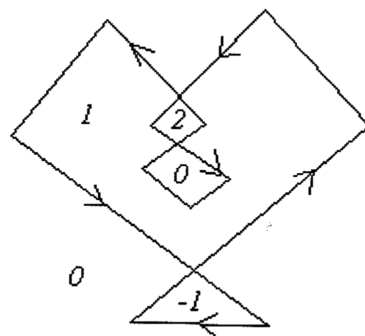


Figure 6.2.1

Given a non-simple polyhedron, the simplification algorithm proceeds by extracting all the regions of the polyhedron that have a positive winding number with respect to the polyhedron. Thus we are extracting all the 'solid' parts of the solid.

Therefore, if the result of applying the algorithm is non-simple, then the simplification algorithm can be applied to obtain a simple polyhedron. Note that the simplification process does not introduce any new planes, and so rounding is not required afterwards.

6.3 Ensuring Containment

In this section we would consider the containment issues for R (the case for R* is similar). We first conjecture that in cases in which the result of the algorithm is simple, the rounded polyhedron is contained within the original.

A justification of this is given as follows. We start with any simple polyhedron that is obtained as a result of applying the algorithm. Figure 6.3.1 illustrates the rounded (defined by the narrower lines) and the original (defined by the darker lines) polyhedrons. As the rounded polyhedron is simple, the normals of the planes are as shown. From the direction of the normals, we see that any vertex of the original polyhedron would have to lie in the region bounded by the dotted lines. This results in the structure that must contain the rounded polyhedron.

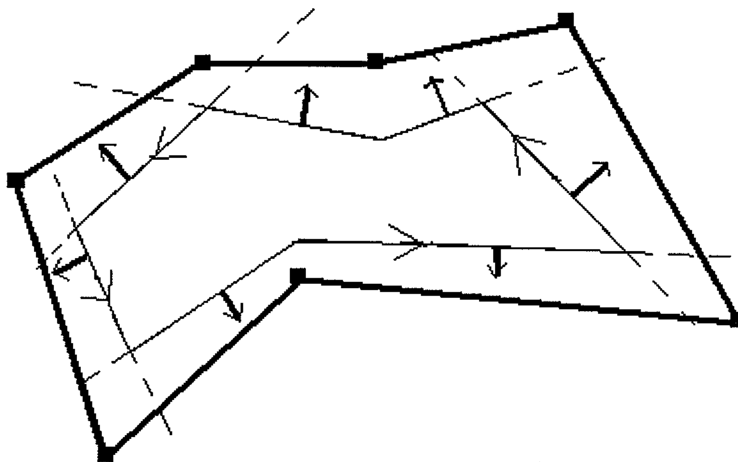


Figure 6.3.1

The problem with this justification is that it is incomplete. This is because it is not proved that the original planes must lie on the side of the rounded plane indicated by the normal (preserving the topological information may invert the direction of the normals after the rounding). This requires a deeper topological and geometrical analysis.

Similarly, in the case of a result that is non-simple, we conjecture that the simplification process described above would ensure containment. We justify this by a process of logical induction rather than deduction – it is true for all examples we have considered, and we could not find a counterexample. The examples are illustrated in figure 6.3.2, figure 6.3.3 and figure 6.3.4 (the original polyhedron is defined by the dark lines, and the rounded one is defined by the lighter lines).

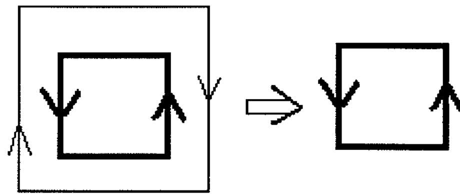


Figure 6.3.2

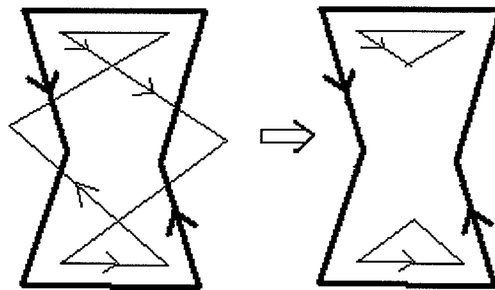


Figure 6.3.3

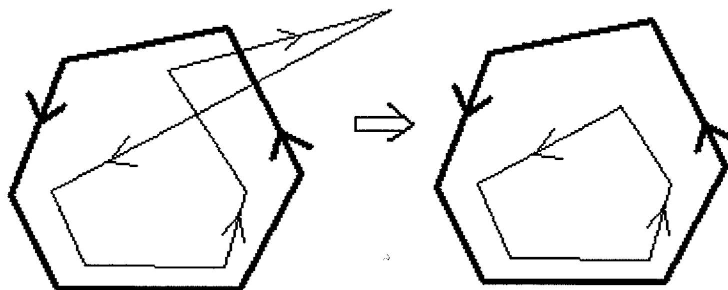


Figure 6.3.4

7. Conclusion

The algorithm presented in this report provides us with a method of polyhedral rounding in the robust framework described in [1]. Given any general polyhedron, that may have irrational coefficients, it can be represented as an interval bounded by two rational polyhedrons. Successive operations on these polyhedrons, that would use exact arithmetic to ensure validity, would result in unbounded growth of the numerical data. At this stage the rounding algorithm described here can be performed to round the inner polyhedron to the inside and the outer one to the outside, thus maintaining the validity of the interval.

We have analysed two methods for rounding the plane normal. The first is aimed at optimising speed, and the second places emphasis on accuracy. In retrospect, the first method turns out to be more preferable than the second, both in terms of speed and accuracy of the overall process. This is because although the rounded normal may be closer using the second method, we saw that the error due to 'shifting' turned out to be greater in this case.

In the last chapter we discussed special cases in which the algorithm may give results that violate the integrity of the polyhedral interval (containment issues) and/or the condition that the result should be simple. These cases can be avoided if the CSG approach is used. Alternatively, we can use Steven Fortune's simplification algorithm. An implementation of this algorithm is a subject of future work. Another topic for further research would be to formally prove that applying the simplification algorithm would also resolve containment issues.

8. Acknowledgements

Special thanks to my supervisor Professor Abbas Edalat, and Ali Khanban for their tireless support throughout the duration of this project.

I would also like to thank Thanos Nikolopoulos, Marko Kerznaric, Ian Moor, and Steven Fortune for their help.

9. Bibliography

[1] Abbas Edalat, Andre Lieutier, "Foundation of a computable Solid Modelling" To appear in Theoretical Computer Science. Extended abstract in Proceedings of ACM's Symposium on Solid Modelling 99, ACM, 1999.

[2] A.Nikolopoulos, "Boolean Operations on Rational Polyhedra", 2001.

[3] Christoph M. Hoffman, "Geometric & Solid Modelling: An Introduction" 1989, Morgan Kauffman Publishers.

[4] The GMP Website,
http://www.swox.com/gmp

[5] "Writing the equation of a line in three dimensions",
http://www.netcomuk.co.uk/~jenolive/vect17.html

[6] S. Fortune, "Polyhedral modeling with multiprecision integer arithmetic", Computer-Aided Design, 29(2), pp 123-133, 1997.

[7] 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.

[8] 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.

User Guide

Special Software required:

Wish 8.3 (for the GUI)
Open GL (for viewing)

The source code is available at ~mr399/project/code. The program is run with the command "wish8.3 gui.tcl"

Input File Format

Topology File:

Number of faces(n)

FaceIndex(1)	*no. of surrounding faces*	*direction of normal(+1 or –1)*	*FaceCycle*
FaceIndex(2)	*no. of surrounding faces*	*direction of normal(+1 or –1)*	*FaceCycle*
	.		
	.		
	.		
	.		
FaceIndex(n)	*no. of surrounding faces*	*direction of normal(+1 or –1)*	*FaceCycle*

Where FaceCycle lists the indices of the surrounding faces

Plane File:

List each plane on a new line, and give the four coefficients a b c d.