

Topologically exact evaluation of polyhedra defined in CSG with loose primitives

Raja (P.K.) Banerjee† and Jarek R. Rossignac

Interactive Geometric Modeling IBM, T.J. Watson Research Center, Yorktown Heights, New York 10598

Abstract

Floating point round-off causes erroneous and inconsistent decisions in geometric modelling algorithms. These errors lead to the generation of topologically invalid boundary models for CSG objects and significantly reduce the reliability of CAD applications. Previously known methods that guarantee topological consistency by relying on arbitrary precision rational arithmetic or on symbol-manipulation techniques are too expensive for practical purposes. This paper presents a new solution which takes as input a “fixed precision” regularized Boolean combination of linear half-spaces and produces a polyhedral boundary model that has the exact topology of the corresponding solid. Each half-space is represented by four homogeneous coefficients in fixed precision format (L_a bits for the three direction cosines and L_d bits for the constant term, i.e. the distance from the origin). Exact answers to all topological and ordering questions are computed using a fixed length, $3L_a + L_d + 2$ bits, integer format. This new guaranteed tight limit on the number of bits necessary for performing intermediate calculations is achieved by expressing all of the topological decisions based on geometric computations in terms of the signs of 4×4 determinants of the input coefficients. The coordinates of intersection vertices are not required for making the correct topological decisions and hence vertices and lines are represented implicitly in terms of planes.

Keywords: constructive solid geometry, half-spaces, loose primitives, polyhedra, robust boundary evaluation, solid modelling.

1. Introduction

Solid modelling provides the core technology for design verification and for the automation of the design and manufacturing planning processes'. Solid modelling computations implicitly rely on complete and unambiguous computer representations of 3D r-sets². CSG representations³ are well suited for such purposes, because they do not contain any topological information that may be inconsistent with the geometric data, even when the geometric information is truncated to a fixed precision numeric format. Boundary representations however, unless triangulated, may be ambiguous. For example, a planar face may be implicitly defined by the location of its four non co-planar vertices, and, without further

information, it may be impossible to decide whether a given point lies on that face or not. Furthermore, boundary models may be inconsistent or even invalid⁴. By inconsistent we mean that the topology (i.e., face-edge-vertex adjacency and ordering information) may not correspond to the actual geometry captured in the vertices' coordinates. By invalid we mean that no possible embedding of the topology may exist in 3-D, as for the case of the Klein bottle.

Since CSG models are compact, always valid, and lend themselves well to high-level design and editing operations, they are often viewed as the primary representation manipulated by the user. Because boundary models permit to derive topological information (such as connectivity or feature existence) not directly available from CSG, it is often desirable to evaluate CSG models, i.e. to convert them to equivalent boundary models. This evaluation process entails the computation of new

† Currently at I.B.M. Orlando, FL. and Univ. of Central Florida, Orlando, FL.

geometric entities (the vertices and edges where primitive faces intersect) and the derivation of the model's topology (essentially the edge-loops or vertex-loops that bound faces and the ordering of faces around edges for non-manifold configurations). Boundary evaluation uses geometric computations to distinguish topological configurations⁵⁻⁷. When carried out in floating point format, these computations typically suffer from round-off errors and may produce inconsistent results, yielding inconsistent or even invalid boundary models. Although validity of the boundary representation may be guaranteed by using validity-preserving incremental construction operators⁸, the resulting models may suffer from two drawbacks: (1) the topology captured in the boundary representation may be inconsistent with the topology implicitly defined by the CSG model, and (2) the geometry may be grossly incorrect and imply a different topology. These drawbacks may invalidate any further processing to be performed on the boundary model.

Several attempts to free geometric algorithms from numerical errors are discussed in Section 3, following an overview of our approach in Section 2. Implementation details are provided in Section 4. The appendix contains an example.

2. Overview of the Proposed Approach

In this section, we present an intuitive overview of our approach, organized into the steps of the proposed design process and of the associated boundary evaluation algorithm.

2.1. Interactive Design with Loose Primitives

In the proposed computer-aided design scenario, designers or application programs construct and edit standard CSG models using the conventional range of parameterized primitive solids and rigid body transformations. By editing the dimensions parameter and the rigid motions of the CSG hierarchy, the shape of the resulting solid may be changed. Direct rendering techniques for CSG may be used to preview the result prior to boundary evaluation (see Rossignac and Wu⁹ for example and references). Conventional CAD systems do not guarantee that the topology of the resulting boundary model will correspond to the topology of the solid represented by CSG for several reasons: (1) rigid body motions are approximated by floating point matrices, (2) curved primitives are often tessellated prior to boundary evaluation, (3) the boundary evaluation algorithms are not reliable. Hence, even in conventional CSG modellers, the shapes, dimensions, positions, and orientations of the primitives used by the merging algorithms are only an approximation of the intended parameters. In the work reported here, this approximation goes one step

further: the topology of the primitives may also be altered. For example, at the apex of a truncated cone the approximation process described below may create small new edges. Thus, designers should keep in mind that they are designing with "loose" primitives whose exact dimensions, position, and also topology are not fixed.

2.2. Fixed Precision Approximation of Primitives

All common primitive solids used in CAD systems may be approximated to any desired accuracy with simple Boolean combinations of linear half-spaces. Cubes, spheres, cylinders, and truncated cones are intersections of half-spaces whose bounding planes support tessellation facets of the bounding surfaces. Tori and other rotational sweeps of 2D contours may also be expressed in CSG form¹⁰, and approximated by simple combinations of linear half-space. Linear half-spaces are represented (as in Slegihara¹¹) by the four floating-point coefficients of their implicit equations. We approximate this representation by normalizing and rounding off the coefficients of the half-spaces to a L_a -bit-long fixed-point format for the three cosine coefficients and to a L_d -bit-long fixed-point for the constant term (the distance to the origin).

The choice of the origin and of the unit for distance permits to optimize the error induced by this round-off. L_a and L_d may be chosen so as to provide the best performance/accuracy compromise and depend on the hardware/software support for efficient long integer operations.

The geometric and topological errors possibly introduced by this approximation are difficult to measure or even bound. However, most users gracefully tolerate errors of similar nature introduced in CSG and in other modelling schemes by applying sequences of rigid body transformations, each approximated by a 3×4 fixed precision matrix, to the geometric entities (planes, lines, or points) represented in the modeller. (Difficulties in preserving model consistency after transformations has been discussed in¹².)

Furthermore, the rounding of normalized plane coefficients in general snaps the normals of almost parallel planes to the same approximate normal and the constant terms of almost coincident planes to the same distance. As a consequence, face/face singularities intended by the designers that may have been lost by applying approximated transformation matrices are often re-established. However, it should be noted that singularities involving primitives edges or vertices are in general not preserved during this approximation.

2.3. Setting-up the Evaluation Tree

The result of the first phase is a CSG tree obtained by replacing each primitive in the original CSG tree with a subtree defining the primitive and approximation in terms of planar half-spaces. The leaves of the expanded tree will hold vectors of discrete values. A list of the half-spaces is constructed, each pointing to a leaf of the new CSG tree. This list is compressed by identifying half-spaces with coplanar boundaries and by combining them. The compressed list entries describe oriented planes. To each plane is associated a list of references to tree leaves. Each reference is marked with an orientation flag specifying whether the leaf half-space is on the inner or the outer side of the plane (with respect to the orientation of normal to the plane).

The CSG tree is used (as explained in the following subsection) to repeatedly evaluate in parallel a vector of Boolean (binary) values, each value corresponds to a sector of an edge neighborhood⁷. Primitive value vectors will be associated with leaves, and the Boolean operations associated with the CSG operators will be applied in parallel to the corresponding vector entries. The resulting vector will define the edge-neighborhood for a particular edge-segment.

2.4. Generate and Test Edges

The CSG evaluation (or “merge”) algorithm follows the approach described in Requicha and Voelcker⁷. Tentative edges (infinite lines) are generated as intersection of pairs of nonparallel planes. Then each infinite line is split into segments by all other planes that intersect it at discrete points. Each segment is tested for its contribution to the solid’s boundary by evaluating its “edge-neighborhood”. The edge splitting and neighbourhood-evaluation are done as follows for each edge.

2.5. Planes Ordering

All planes (of the list) except the two used to form the line (tentative edge) are classified into three categories: disjoint from the line, containing the line and splitting the line.

All the half-spaces associated with disjoint planes are used to modify the initial values of the evaluation vector for that particular line.

Planes containing the line are inserted in a radial ordering of planes around the line. They will split the line’s edge-neighborhood into radial sectors. Each sector will be associated with an entry of the evaluation vector.

Planes splitting the line are inserted in a partial ordering in terms of their intersection points with the line. At the end of this process, they split the line into connected

tentative edge-segments separated by vertices that lie on one or more splitting plane.

All the above ordering is performed without numerical error and without computing explicit representation of the line or of its intersection points with the splitting planes. Comparisons necessary for the ordering are expressed in terms of signs of determinants using the approximated coefficients of the plane equations.

2.6. Edge Segment Testing

The entries in the evaluation vector are initialized during the above splitting process for the first (semi-infinite) segment along the line. When dealing with bounded CSG models, the neighbourhood of that segment should always evaluate to empty, i.e. the resulting evaluation vector should contain all zeroes implying that all neighbourhood sectors are out of the solid.

As we move to the next segment along the line, we traverse one or more splitting planes (their list is associated with the traversed vertex). Each plane points to a list of half-spaces, i.e. leaves whose neighbourhood evaluation vector must be flipped. The neighbourhood, i.e. the vector valued CSG expression is re-evaluated. If the resulting evaluation vector is not uniform (all zeroes or all ones), then the edge segment with the associated edge-neighborhood information (half-spaces bounding the non-empty sectors) is added to the resulting edge-list.

2.7. Boundary Graph Reconstruction

The ordering and relative orientation information captured in the resulting edge list may be used to reconstruct the face-edge adjacency and orientation information and an SGC structure is constructed for the vertex-edge-face adjacency graph¹³. Note that at this point, the “extents”, i.e. geometric entities that support the edges and vertices, are represented in implicit form as intersections of two or three planes. The SGC representation may be used to derive further topological partitioning of the objects into connected solid components.

Vertices are evaluated by solving the linear systems of three plane equations in floating point. These approximate values are merely used for graphics and numeric computations.

The algorithm described above produces edge neighbourhoods from which a boundary model, that has the exact topology of the approximated CSG model, can be constructed.

3. Related Work

The problem of numerical errors in geometric computations has received considerable attention in recent years.

We summarize some key techniques below. Most approaches are surveyed in more details in¹⁴.

3.1. Intelligent Floating Point Calculations

Considerable improvements to techniques based on numerical tolerances have been achieved by intelligent strategies that (1) avoid many inconsistent decisions^{15,16} and (2) chose the most accurate technique when several are available for computing the same result^{17,18}. Because it is not always possible to guarantee global consistency with a prescribed geometric accuracy, and because even the most accurate floating point computations may lead to inconsistent results, these schemes may fail to produce a valid and accurate approximation of the boundary. However, the range of critical situations that cannot be consistently handled by reported implementation is extremely small and thus perfectly acceptable in most design environments. A more problematic issue concerns the lack of consistency of the algorithms with the expected algebraic properties of the operators they implement. For example, it may be important for applications to assume that $A \cap B$ implies $B \cap A$. Yet, many strategies that avoid inconsistent decisions may fail to meet this requirement.

3.2. Error Bounds and Interval Arithmetic

Other techniques focus on the control of round-off errors in numeric representations. Interval arithmetic applications to geometry are discussed in Mudur and Koparkar¹⁹. New arithmetic operators that guarantee the accuracy of the floating point representation are provided in Ottmann et al.²⁰. Error bounds on intermediate computations, also called “uncertainty interval”, are used to detect and correct numerical uncertainties^{21,22}.

The idea of numeric intervals has been extended to more explicit geometric error bounds^{23,24}, which are used to force geometric singularities. An adaptive resolution technique is used in Bruderlin²⁵ to detect possible inconsistencies with a given resolution and to re-execute the computations with a higher precision.

3.3. Topologically Valid Geometric Approximations

Some approaches may be classified as performing topologically consistent geometric round-offs of intermediate results during the geometric intersection calculations^{26,27}. The computation of the intersection of line segments in the plane²⁸, and of polygons²⁹ are good examples. The segment’s end points are snapped to an integer grid or to nearby lines, both at the input to the algorithm and during calculation as new intersection points are found that break segments into smaller ones. The result is the intersection of polylines that approximate the original line segments, but have the same

topology. A similar approach is used in Milenkovic³⁰ for computing the intersections of polygons on two dimensions, but instead of vertex snapping, line coefficients are rounded-off to fixed precision. The idea of producing a consistent representation that is “close” to the correct one was also used by Fortune for robustly computing convex hulls and triangulations in two dimensions³¹.

3.4. Exact Numeric Computations

As an extreme, extended precision computations have been used³² to perform Boolean operations exactly on a restricted class of polygons and polyhedra using fixed precision representation of line and plane coefficients¹¹. This approach is similar to ours, because it also assumes that the input planes are represented exactly with fixed precision. One difference with our approach lies in the fact that Sugihara and Iri suggest increasing the precision for the primitives’ bounding plane equations as these primitives are manipulated by the user so as to preserve the primitive’s topology and consistency with the vertex geometry. We achieve a fixed low bound on the length of the representation needed for all internal calculations by relying on the CSG representation. Another difference is that we do not use extended precision to compute or represent vertices, but only to compute the exact answers to all topological questions. The coordinates of the vertices of the resulting solid are approximated a posteriori using floating point calculations. They are not used for establishing the topology of the boundary, nor to store intermediate results.

The idea of using exact evaluation of the sign of determinants to guarantee robustness is central to the work reported here. It has been previously applied to achieve robust triangulation using rational coefficients³³. For simplicity and performance, we restrict our implementation to integer coefficients.

4. Algorithm

Table-1 shows a high level pseudo code for the proposed algorithm. Here P_a denotes plane A, L denotes the infinite line (tentative edge) of intersection of two planes, I denotes a point at infinity on line L, S denotes the linear list of segment planes partitioning L and C denotes circular list of planes containing L. Note that the location of I is determined by the first segment plane P_s^1 , as I is always assumed to be outside the plane P_s^1 .

We first define precisely the concepts and representations used in the algorithm.

4.1. Approximation of the Planes

A plane P_i in 3-dimensions is represented by four coefficients a_i, b_i, c_i, d_i and is defined as

Algorithm-1

```

1. Gather all unique planes of the CSG tree
   in the plane set
2. for (each unordered pair  $(P_a, P_b)$ 
   in the plane set) do
3.   if (pair  $(P_a, P_b)$  not processed) and
     ( $P_a$  not parallel to  $P_b$ ) then {
4.      $L = P_a \cap P_b$ 
5.     if ( $L$  is finite) then {
6.       /* at least one Segment Plane  $P_s^1$  exists */
7.       Initialize  $C = \{ P_a^+, P_b^-, P_a^-, P_b^+ \}$ 
8.       Mark the leaves of  $P_a$ 
9.       and  $P_b$  as ON in CSG tree
10.      Initialize  $S = \{ P_s^1 \}$  /* first
11.      sectioning plane */
12.      Mark the leaves of  $P_s^1$  as OUT
13.      in CSG tree /* by definition of I */
14.      edgeOrientation = sign of the 3 by 3
15.      determinant formed by
16.      coefficients of planes  $P_a, P_b, P_s^1$ 
17.      for (each  $P_j$  except  $P_a,$ 
18.       $P_b$  and  $P_s^1$  in plane set) do
19.        if ( $L \cap P_j \neq \emptyset$ )
20.          then { /*  $P_j$  intersect  $L$  at a vertex */
21.            Refine  $S$  using  $P_j$ 
22.            Mark the nodes of  $P_j$  as
23.            IN or OUT in the CSG tree
24.            based on the classification
25.            of  $I$  with respect to  $P_j$  }
26.          else
27.            if ( $L$  in  $P_j$ ) then
28.              { /* plane  $P_j$  contains line  $L$  */
29.              Mark pairs  $(P_j, P_k)$  as
30.              processed for all  $P_k$  in  $C$ 
31.              Refine  $C$  using  $P_j$ 
32.              Mark the leaves of  $P_j$ 
33.              as ON in the CSG tree }
34.            else /*  $P_j$  is parallel to line  $L$  */
35.              Mark the nodes of  $P_j$  as
36.              IN or OUT in the CSG tree
37.              based on the classification of
38.               $I$  with respect to  $P_j$ .
39.            endif
40.          endif
41.        enddo /* for each  $P_j$  .. */
42.      /* at this stage two ordered lists  $S$  and  $C$  have been
43.      built, each plane  $P_s$  in  $S$  refers to a tentative
44.      vertex of the solid. Next section evaluates the tree */
45.      for (each tentative vertex in  $S$ ) do
46.        Evaluate root node value of CSG tree
47.        if (root is ON) then {
48.          Evaluate the neighbourhood half-spaces
49.          Evaluate edge face neighbourhood.
50.          Insert the edge in SGC }
51.        endif
52.      Flip the classification for
53.      leaves of  $P_s$  in CSG tree
54.      enddo /* for each tentative vertex ... */
55.      endif /*  $L$  is finite */
56.      endif /* pair  $P_a, P_b$  ... */
57.    enddo /* for each unordered pair ... */
endproc

```

Table 1: Algorithm for Robust Boundary Evaluation of Polyhedra defined in CSG

$a_i x + b_i y + c_i z + d_i = 0$, where at least one of the three coefficients a_i, b_i, c_i is not zero. The normal (a_i, b_i, c_i) defines an orientation. A linear **half-space** is bounded by a plane and has a one bit orientation flag indicating which side of the plane is the interior of the half-space. If the flag is a one, the normal points towards the interior. A plane can thus support two half-spaces, one on each side.

Sugihara outlines four fundamental approaches for approximating the real coefficients of a plane equation by fixed precision rational numbers¹¹. These heuristic methods can be classified as the Naive method, the Exhaustive search method, the Continued fraction method and the Basis reduction method. He recommends use of a hybrid method comprising of the Basis reduction method, Naive method and the Continued fraction method for the best possible result in approximating lines for 2-dimensional polygons. In our implementation we have used the naive method for simplicity.

Each plane is represented by four homogeneous coefficients in fixed precision format (L_a bits for the three direction cosines and L_d bits for the constant term). Exact answers to all topological and ordering questions are computed using a fixed length, $3L_a + L_d + 2$ bits, integer format. The above size is determined as follows.

The 4 by 4 determinant has 24 terms of the form $a_i b_j c_k d_l$. The product $a_i b_j$ needs $2L_a - 1$ bits for exact representation, $a_i b_j c_k$ needs $3L_a - 2$ bits and $a_i b_j c_k d_l$ needs $3L_a + L_d - 3$ bits. The sum of 24 such terms needs at least 5 more bits. The number of bits needed for exact representation of a 4 by 4 determinant is $3L_a + L_d + 2$ bits.

If the cosine terms of the planes are represented using integers of size L bits and the constant coefficient is represented using integers of size $2L$ bits, the result of an exact computation for a 4 by 4 determinant will be $5L + 2$ bit long. Using the mantissa of a double precision IEEE floating point number (53 bits) to represent this exact result, the maximum size of L is 10 bits including the sign bit. Consequently, if we normalize the cosine coefficients to be between -2^9 and 2^9 and the constant term d_i to be between 0 and 2^{20} , we can use double precision floating point to do all computations.

4.2. Creating the Plane Set

After the planes coefficients are approximated by fixed precision integer format, this step is used to eliminate any identical/coincident planes. This is done by eliminating any common factor from the four coefficients of the plane, by computing the greatest common divisor (GCD) of the coefficients and then dividing each coefficients by the GCD. Then the planes are lexico-

graphically sorted using their coefficients. The sorting step eliminates any coincident planes.

4.3. Tests on Planes

Two planes P_1 and P_2 are **parallel** iff the three determinants

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \quad \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} \quad \begin{vmatrix} c_1 & a_1 \\ c_2 & a_2 \end{vmatrix}$$

are zero. If all six cosine terms are non-zero it is sufficient to check for nullity of any two of the above three determinants.

Two *parallel* planes P_1 and P_2 are **identical** iff the three determinants

$$\begin{vmatrix} a_1 & d_1 \\ a_2 & d_2 \end{vmatrix} \quad \begin{vmatrix} b_1 & d_1 \\ b_2 & d_2 \end{vmatrix} \quad \begin{vmatrix} c_1 & d_1 \\ c_2 & d_2 \end{vmatrix}$$

are zero.

A line of intersection of two planes P_1 and P_2 is **parallel** to a third plane P_3 iff the determinant

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = 0.$$

A line of intersection (L) of two planes P_1 and P_2 is **contained** in a third plane P_3 iff, L is *parallel* to plane P_3 and any point on the line L lies ON the plane P_3 .

4.4. Side Test

Given a vertex V represented by the intersection of three planes P_1, P_2 and P_3 ; and a plane P_4 let:

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

If $J = 0$ then vertex V is ON P_4 otherwise V is on the positive side of P_4 if and only if J and D have the same signs.

4.5. Linear Ordering of Planes

Sectioning planes of type P_s subdivide line L into a set of edge segments. All vertices formed by the intersections of L and planes of type P_s are linearly ordered along L using insertion sort and decisions based on the exact computation of determinants.

The first step of the ordering of vertices along an unbounded line L is to find two *sectioning planes* which intersect the line L at two distinct vertices and order

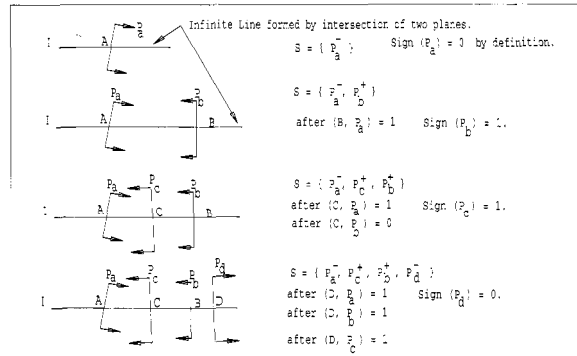


Figure 1: Linear ordering of a sectioning plane P_j in S using Boolean functions $\text{after}(V, P_j)$ and $\text{sign}(P_j)$.

them properly with respect to the assumed location of origin (point at infinity). Next any other sectioning planes intersecting the line L at a vertex are ordered by using insertion sort and two Boolean functions as described below (see Figure 1).

Let S be an ordered list of planes, and including a relative orientation sign bit with respect to the orientation of the first plane in the list.

Assume we have an unbounded line $L (P_1 \cap P_2)$ and a plane P_a intersecting L at point A. We choose as origin of the line L the point at infinity I, on the negative side of P_a (as such plane A is denoted as P_a^- in the ordered list S). The sign bit for P_a in S is therefore set to Boolean value FALSE.

Point B ($L \cap P_b$) is in between points I and A if and only if B is on the negative side of P_a . It is possible that B is on P_a , then points A and B are identical (we have a vertex which is the intersection of four planes). Another test on point I with respect to plane P_b determines its sign in S (if I is on positive side of plane P_b it will be represented as P_b^+ otherwise by P_b^-). Hence two sectioning planes are ordered in the list S.

The above facts can be generalized by defining two functions as follows:

Let $\text{after}(V, P_j)$ be a Boolean function indicating if point V is after plane P_j in the ordered list S, and let S_j denotes the sign of P_j in S. Then

$$\text{after}(V, P_j) = (V \in P_j) \oplus S_j.$$

The expression $(V \in P_i)$ return a Boolean value TRUE when point V is in positive side of plane P_j .

Also if point V is found to be before P_j in the ordered list S then a Boolean function $\text{sign}(P_v)$ is defined as:

$$\text{sign}(P_v) = (J \in P_v) \oplus S_j.$$

We can find the exact location of a given a new plane

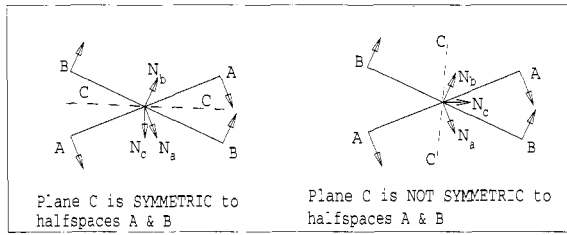


Figure 2: Two cases illustrating when a Plane P_c is symmetric and un-symmetric with respect to a pair of planes P_a and P_b .

P_k in the ordered list $S = P_a^-, P_b^+, \dots$, by computing the function *after* (K, P_j) for planes $P_j, j = a, b, \dots$ etc., in the ordered list S, until a plane P_l is found where K is before plane P_l . Plane P_k is then inserted in the list S before plane P_l . If no such plane P_l is found then P_k is inserted at the end of the list.

The sign of plane P_k can be found from the second function using sign of plane P_l .

4.6. Circular Ordering of Half-spaces

The planes containing L (P_c^1, P_c^2 , etc.) and planes defining L (P_i, P_j) are circularly ordered around the line L. These planes can have two possible circular orderings one in clockwise and the other in counterclockwise direction (see Figure 3) with respect to the orientation of L implied by the first section plane.

Note that for every edge, there will be at least a circular order of two half-spaces, the half-spaces whose bounding planes (P_i, P_j) define the unbounded line L. All planes of type P_c and planes P_i, P_j are circularly ordered using decisions based on exact computation of determinants.

The circular ordering is also done using insertion sort, but here we first find if a given plane is symmetric with respect to two circularly ordered half-spaces. Given two planes P_a and P_b , a plane P_c is **symmetric** to P_a and P_b if for any point on P_c the classification with respect to planes P_a and P_b are identical (see Figure 2).

Let P_s^1 be the first sectioning plane for the infinite line of intersection of planes P_a and P_b and plane P_a^1 is a plane which is parallel to P_a and at a small distance from P_a (formed by adding a 1 to the constant term of P_a). We implicitly define a point X as the intersection of planes P_s^1, P_c and P_a^1 . This point X is on P_c but not on planes P_a and P_b . Plane P_c is symmetric to planes P_a and P_b if and only if point X lies on the same side of planes P_a and P_b i.e. the classification of X with respect to planes P_a and P_b are identical. The procedure

```

procedure-2
IsSymmetric( $P_a, P_b, P_c, P_s^1$ )
begin
1. define  $X = (\text{shifted } P_a) \cap P_c \cap P_s^1$ 
2. return ((  $X \in P_a$  ) and (  $X \in P_b$  ))
end
    
```

Table 2: Procedure for finding if a given plane is symmetric to two planes.

in Table-2 is used to determine if plane P_c is symmetric to half-spaces P_a and P_b :

Another method to identify if a plane P_c is symmetric to planes P_a and P_b using vector algebra is as follows; Let N_a, N_b and N_c be the vectors identifying the normal to planes P_a, P_b and P_c respectively. Plane P_c is symmetric to P_a and P_b iff $((N_a \times N_b) \times N_c) \cdot N_a$ and $((N_a \times N_b) \times N_c) \cdot N_b$ have same sign. If subscripts 1, 2 and 3 are used to denote the cosines of the normals N_a, N_b and N_c respectively then the expression $((N_a \times N_b) \times N_c) \cdot N_a$ can be written as the determinant:

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

The other expression $((N_a \times N_b) \times N_c) \cdot N_b$ can also be written similarly in a determinant form. Note that to exactly compute the sign of this determinant we will need $(2L_a - 1) + 2L_a - 2 + 3 = 4L_a$ bits, integer format, where L_a bits are needed to represent each of the cosine terms a_i, b_i, c_i for $i=1,2,3$.

4.7. Edge Neighbourhood

When an edge is classified as ON, its neighbourhood is available. It is simply represented as a string of bits, each bit specifying whether the neighbourhood wedge of two consecutive planes (see Figure 3) is in or out of the solid.

The **segment neighbourhood vector** (abbreviated as **SNV**), comprising of an array of bits, is used to represent the edge neighbourhood for an edge. Two successive neighbourhood planes in the circularly ordered list bound a sector of these neighbourhoods. Note that the pair of planes appears twice. Each bit of the segment neighbourhood vector has a one to one correspondence with each of these sector intervals.

A 1 in the SNV bit signifies presence of material in that sector around the edge, and a zero indicates an empty sector. For example a segment which is inside the solid composition is represented by a SNV with all bits set to one, and a segment which is outside the

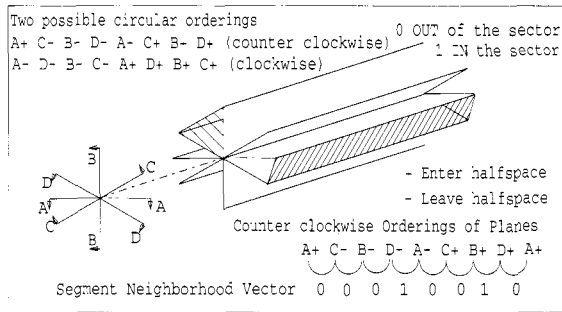


Figure 3: Circular ordering: Four half-spaces passing through an edge, the two possible representation of their circular order and the segment neighbourhood vector for counterclockwise order.

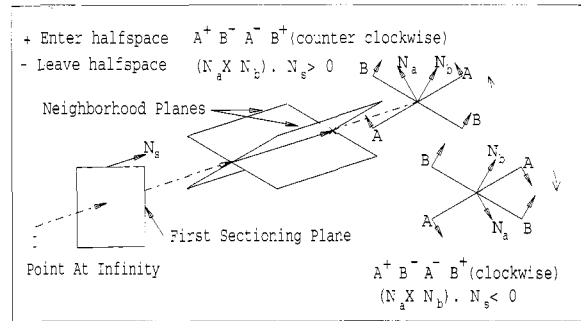


Figure 4: Relationship between edge orientation and circular order.

composition represented by a SNV with all bits set to zero.

To compute the edge neighbourhood of a line $L (P_i \cap P_j)$, we start by classifying the point at infinity on the infinite line L with respect to the half-spaces of the CSG tree. The point at infinity is classified as ON for all half-spaces of type P_c and planes P_i , and P_j , since these planes contain the unbounded line L and hence the point at infinity. The number of planes containing the line defines the size of the SNV. For example, consider the case where four planes P_a, P_b, P_c and P_d contains the line defined by $P_a \cap P_b$. In this case, the size of SNV is 8 bits and the leaf of CSG tree associated with plane P_a will be initialized as "11110000", the leaf of CSG tree associated with plane P_b will be initialized as "11000011", the leaf of CSG tree associated with plane P_c will be initialized as "10000111" and the leaf of CSG tree associated with plane P_d will be initialized as "11100001" respectively.

The point at infinity is classified against all other half-spaces as IN or OUT. Every point on the unbounded line L has the same classification with respect to the half-spaces which are on planes of the first type, P_d , since the line L is parallel to all such disjoint planes.

4.8. Tree Evaluation

The classification of the edge segment is determined by evaluating the CSG expression on these SNV's and computing the resulting vector. If the final computed SNV contains only ones, the edge segment is classified as IN, if it contains all zeroes then the edge segment is classified as OUT. However, if the final vector contains both ones and zeroes then the edge segment is classified as ON and we store the edge neighbourhood (i.e. the half-spaces which form the sectors) by using the circular ordering of the half-spaces.

4.9. Vertex Traversal

Once a segment's neighbourhood is evaluated, we move to the next tentative vertex, which points to a set of sectioning planes (entries in the linearly ordered list of planes S). For each of these planes we traverse its list of leaf references and for each reference, flip all the bits of the vectors at the leaf. Then we re-evaluate the tree, and if the resulting segment is ON, we add it to the edge list with the appropriate neighbourhood information.

After evaluation of edge neighbourhood of a tentative edge, if the tentative edge is found to be ON then it is compared with the neighbourhood of the prior tentative edge segment which was ON. If the neighbourhood planes are same for both tentative edges, then the two tentative edges are merged by eliminating the common segment plane. This avoids breaking a single continuous edge into smaller pieces with identical neighbourhoods.

4.10. Edge Orientation and Neighbourhood

In our algorithm, we process tentative edges on an infinite line ($L = P_a \cap P_b$), starting from the point at infinity on the line and proceeding in the direction of the normal to the first sectioning plane (i.e. in the direction of the arrow as shown in Figure 4). We also initialize the circular order based on the two planes P_a and P_b (step 6 in algorithm 1). As mentioned earlier, the circular order can be counter clockwise or clockwise based on the relative orientation of planes P_a and P_b . If sign of the vector expression $(N_a \times N_b) \cdot N_s$ is positive then the circular order is counter clockwise looking in the direction of edge orientation; otherwise the circular order is clockwise. N_a, N_b and N_s are vectors in the direction of the normal to the positive side of the planes P_a, P_b and the first sectioning plane. The above sign can be

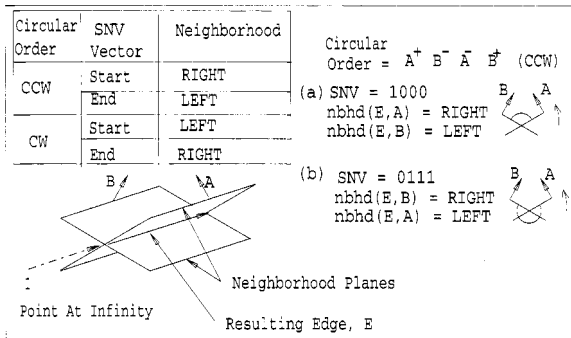


Figure 5: Finding left and right neighbourhoods of an edge from the SNV and circular order.

computed by evaluating the determinant

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

where subscripts 1,2 and 3 denotes the direction cosines for the planes P_a , P_b and the first sectioning plane respectively.

Any boundary representation (SGC) will need left and right relative neighbourhood information for face-edge and edge-vertex pairs [13]. The edge-vertex neighbourhood of a regular solid can be **left**, **right**. The edge-vertex neighbourhood depends on the order an edge is traversed and the linear order of sectioning planes gives us that information. For a resulting edge of a solid, the vertex associated with the sectioning plane which is closest to the point at infinity (i.e. encountered first in the direction of edge orientation) has a neighbourhood of *left*. The other vertex of the resulting edge has the neighbourhood *right*.

The face-edge neighbourhood of a regular solid can be **left**, **right** or **full**. In our algorithm edges with *full* neighbourhood (whose SNV's are all ones) are computed but not selected for final output. So the face-edge neighbourhood can only be *left* or *right*. Once the direction of circular order is known, it can be used to compute the left, right neighbourhood of the edge with respect to the two faces if the edge was classified as ON. In this case the SNV will have zeroes and ones. The ones in the SNV indicate the sector which is part of the neighbourhood. The neighbourhood of the edge with respect to the starting (beginning) face of the SNV sector will be RIGHT if the circular order was counter clockwise (refer Figure 5). The table in this figure illustrates how the neighbourhood of the edge is computed with respect to the faces of the sectors. For example, for a counter clockwise circular order A^+ , B^- , A^- , B^+ if the SNV is "1000" for an edge E, then the **neighbourhood(E,**

P_a) = right and the **neighbourhood(E, P_b)** = left (see Figure 5). If the SNV is "0111" for an edge E, then the **neighbourhood(E, P_b)** = right and the **neighbourhood(E, P_a)** = left.

5. Conclusion

Our work is based on the use of error-free computations to construct the exact topologies of CSG solids obtained by approximating the CSG primitives with Boolean combinations of planar half-spaces with integer coefficients of fixed precision. We reduce all the geometric tests necessary for boundary evaluation to questions regarding the relative ordering of implicit entities. Specifically, we use implicit representations for edges and vertices, storing them as intersections of two or three planes. Each half-space is represented by coefficients of the plane equation and an orientation flag. We use linear and circular ordering queries that operate on this implicit representation. All queries are answered by testing the sign or nullity of 4 by 4, 3 by 3 and 2 by 2 determinants. Special purpose hardware is currently being designed to evaluate efficiently such determinants of fixed precision. The algorithm reported here was tested on a software simulator.

The major contribution of this paper is the outline of a detailed algorithm that implements a complete boundary evaluation of CSG models of polyhedral solids, including singular cases, and produces the exact topological boundary for the approximated CSG solid.

References

1. A. A. G. Requicha and J. R. Rossignac, "Solid Modeling and Beyond", *IEEE Computer Graphics and Applications*, **12(5)**, 31-44, (September 1992).
2. R. B. Tilove and A. A. G. Requicha, "Closure of Boolean Operations on Geometric Entities", *Computer-Aided Design*, **12(5)**, 219-220, (September 1980).
3. A. A. G. Requicha, "Representations for rigid solids: Theory, methods, and systems", *ACM Computing Surveys*, **12(4)**, 437-464, (December 1980).
4. C. M. Hoffmann, "The problem of accuracy and robustness in geometric computation", *IEEE Computer*, **22**, 31-42, (1989).
5. M. Maentylae, "Boolean operations of 2-manifold through vertex neighborhood classification", *ACM Trans. On Graphics*, **5(1)**, 1-29, (1986).

6. D. Laidlaw, W.B. Trumbore, and J.F. Hughes, "Constructive solid geometry for polyhedral objects", *Computer Graphics*, 20(4), 161–170, (1986).
7. A. A. G. Requicha and H. B. Voelcker, "Boolean operations in solid modeling: Boundary evaluation and merging algorithms", *Proc. IEEE*, 73(1), 30–44, (January 1985).
8. M. Maentylae, "A note on the modeling space of euler operators", *Computer Vision, Graphics, and Image Processing*, 26, 45–60, (April 1984).
9. J. R. Rossignac and J. Wu, *Correct Shading of Regularized CSG Solids Using a Depth-Interval Buffer*, Springer-Verlag, Eurographics Seminars, Berlin, (1990).
10. D. L. Vossler, "Sweep-to-csg conversion using pattern recognition techniques", *IEEE Computer Graphics and Applications*, 5(8), 61–68, (August 1985).
11. K. Sugihara, "On finite precision representations of geometric objects", *Research Memorandum RMI 87-06, Faculty of Engg.*, (1987).
12. V. J. Milenkovic and L. R. Nackman, "Finding compact coordinate representations for polygons and polyhedra", *ACM Symposium on Computational Geometry*, pp. 244–252, (1990).
13. J. R. Rossignac and M. A. O'Connor, *SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries*, North-Holland, Rensselaerville, NY, (September 1989).
14. A. J. K. Stewart, "The theory and practice of robust geometric computation, or, how to build robust solid modellers", *PhD thesis*, (1991).
15. K. Sugihara, "A simple method of avoiding numerical errors and degeneracy in voronoi diagram constructions", *Dept. of Math. Engr. and Instru. Physics*, (1988).
16. K. Sugihara and M. Iri, "Two design principles of geometric algorithms in finite precision arithmetic", *Applied Mathematical Letters*, 2(2), 203–206, (1989).
17. M. Karasick, "On the representation and manipulation of rigid solids", *PhD thesis*, (1988).
18. C. Hoffmann, J. Hopcroft, and M. Karasick, "Robust set operations on polyhedral solids", *IEEE Computer Graphics and Applications*, 9(6), 50–59, (November 1989).
19. S. Mudur and P. Koparkar, "Interval methods for processing geometric objects", *IEEE Computer Graphics and Application*, pp. 7–17, (February 1984).
20. T. Ottmann, G. Thieme, and C. Ullrich, "Numerical stability of geometric algorithms", *ACM Proc. of the 3rd Annual Sym. on Computational Geometry*, pp. 119–125, (June 1987).
21. L. Guibas, D. Salesin, and J. Stolfi, "Building robust algorithms from imprecise computations", *Proc. 1989 ACM Sym. on Computational Geometry*, pp. 198–217, (1989).
22. D. Salesin, "Epsilon geometry: Building robust algorithms from imprecise computations", *PhD thesis*, (1991).
23. M. Segal and C. Sequin, "Consistent calculations for solid modeling", *Proc. 1st ACM Sym. on Computational Geometry*, pp. 25–38, (1985).
24. M. Segal, "Using tolerances to guarantee valid polyhedral modeling results", *Computer Graphics, SIGGRAPH '90*, 24(4), 105–114, (August 1990).
25. B. Bruderlin, "Robust regularized set operations on polyhedra", *Hawaii International Conference on System Sciences*, (January 1991).
26. K. Sugihara, "A robust intersection algorithm based on delaunay triangulation", *CS Dept. Purdue Univ Report 91-011*, (February 1991).
27. K. Sugihara, Ooishi, and T. Imai, "Topology-oriented approach to robustness and its applications to several voronoi diagram algorithms", *Canadian Conference on Computational Geometry*, pp. 36–39, (1990).
28. D. H. Greene and E F. Yao, "Finite-resolution computational geometry", *IEEE Annual Symposium on Foundations in Computer Science*, pp. 143–152, (October 1986).
29. C. Hoffmann, J. Hopcraft, and M. Karasick, "Towards implementing robust geometric computations", *ACM Symposium on Computational Geometry*, pp. 106–117, (June 1988).
30. V. J. Milenkovic, "Verifiable implementations of geometric algorithms using finite precision arithmetic", *PhD thesis*, (1988).
31. S. Fortune, "Stable maintenance of point set triangulations in two dimensions", *Annual Symposium on Foundations in Computer Science*, pp. 494–499, (1989).
32. K. Sugihara and M. Iri, "A solid modelling system free from topological inconsistency", *Research Memorandum RMI 89-03, Faculty of Engg.*, (1989).
33. M. Karasick, D. Lieber., and L. Nackman, "Efficient Delaunay triangulation using rational arithmetic", *ACM Transactions on Graphics*, 10(1), 71–91, (January 1991).

Appendix A: An Example of Finding Edge Neighbourhood

We consider the case of finding the edge neighbourhoods of the regularized difference of two blocks A and B. Block B has the same length as block A but is of different width and height and is rotated about the Z axis and shares an edge with block A. (see Figure 6).

The approximated CSG tree is shown in Figure 8. We number half-spaces in the following order for each block: left side, right side, bottom, top, back and front. We only assign new indices if the half-space is unique. Since both blocks have identical length, two leaves of the CSG sub-tree for the Block B point to the same half-spaces H_5 and H_6 of the Block A respectively. For this problem the plane set contains ten planes corresponding to the ten half-spaces H_1 through H_{10} .

Consider the infinite edge created by the intersection of the top plane of Block B (half-space H_{10}) and the front plane of Block A (half-space H_6), as shown in Figure 7.

The first half-space in the plane set which intersects this infinite line at the point Q, is the half-space iH_1 with plane equation, $x = 0$. H_1 therefore, becomes the first segment plane. The point at infinity I, is assumed to be outside H_1 and hence is on the left side of the line (see Figure 7). The next half-space H_2 intersects the edge at point T, and is placed after H_1 , thus forming the segment list, S.

Half-spaces H_3, H_4, H_7 and H_8 intersect the edge at points U, P, R and S respectively and are inserted in the linearly ordered Segment list.

While inserting these half-spaces in the linearly ordered list, we also check on which side of these half-spaces point I lies, and mark the leaves of the CSG tree using these half-spaces as IN or OUT accordingly (see Figure 8).

The half-space H_5 is the only half-space which is parallel to the infinite edge, hence point Q is classified with respect to H_5 and point I is given the same classification as that of Q. In this case point Q is IN half-space H_5 .

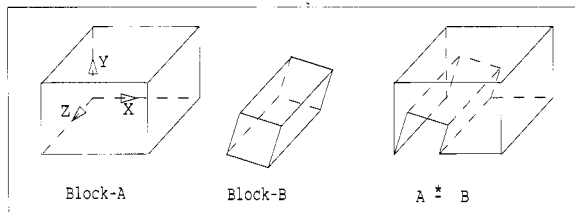


Figure 6: Regularized difference of two blocks: The boundary evaluation of two blocks.

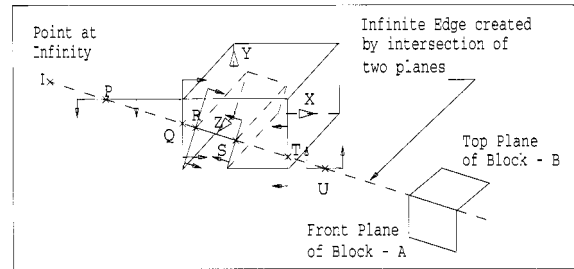


Figure 7: Segments of an edge: An infinite line created by intersection of two planes and its segments created by the intersection of the line with other half-spaces.

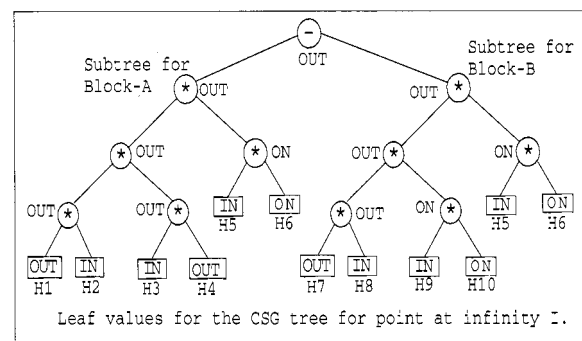


Figure 8: CSG tree for point at infinity: The node values for the CSG tree for the point at infinity I, for the edge in Figure 7. The point at infinity I, and hence the segment IQ is classified as OUT.

The half-spaces H_6 and H_{10} pass through the edge and hence are ordered circularly and the leaves of the CSG tree which point to these half-spaces are marked as ON. A complete CSG tree, with its classification for the point at infinity I, is shown (see Figure 8).

Once all the planes and all the half-spaces have been processed, we can evaluate the classification at the root of the tree by combining the neighbourhood classifications according to the Boolean operator at the parent node. For this example, the root node is evaluated as OUT and hence the segment IQ is rejected.

To reach the next segment on the line, we traverse point P next in the linearly ordered segment list (refer Figure 7), the associated half-space is H_3 , then find the leaf node in the CSG tree which is using this half-space and change the associated classification. In this case the classification was changed from OUT to IN and the neighbourhood for the new segment is evaluated (see Figure 9).

The value at the root node is still OUT, and hence segment PQ is not on the composition. The process is

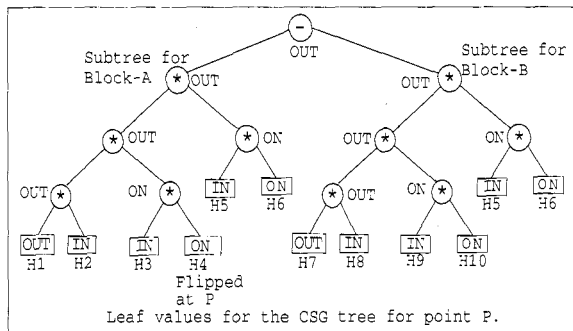


Figure 9: CSG tree for point P: The node values for the CSG tree for the Point P, for the edge in Figure 7. The classification of the leaf node using the half-space at point P was changed from IN to OUT and tree reevaluated.

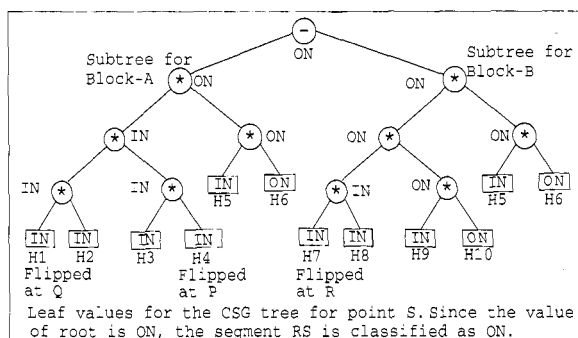


Figure 10: CSG tree for point S: The node values for the CSG tree for the point at R, for the edge in Figure 7. Since points P, Q, R were passed to reach point S, the classifications were changed from OUT to IN for nodes using these half-spaces.

continued until we reach the end of the linearly ordered list. The ON segments are obtained when we visit the half-space H_8 i.e. after traversing point R as shown (see Figure 10).

In the case of this infinite line, none of the half-spaces intersected at the same point on the edge. However, for the edge created by the bottom plane of Block B and front plane of Block A, four half-spaces intersect the line at point R forming a vertex star. (see Figure 11)

In such situations we put in the first half-space encountered in the linearly ordered list, and the other half-spaces are kept in another list of "Same Segments" half-spaces starting from the first half-space. While visiting the first half-space we also visit these list of "Same Segments" half-spaces and change the classification of the leaf nodes which are using these half-spaces.

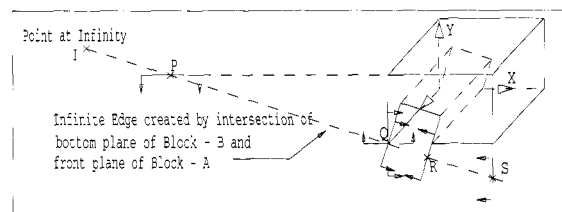


Figure 11: Formation of vertex star: Four half-spaces intersect the edge formed by bottom plane of block B and front plane of block A at point R to form a vertex star.

Next, we consider an unbounded line where multiple neighbourhood planes pass through the line and examine how to compute the neighbourhoods for an edge segment which is classified as ON. To better illustrate this case we consider the line generated by the intersection of bottom plane of Block A (half-space H_1) and the side plane of Block A (half-space H_3). For this line the half-spaces H_1, H_3, H_7 and H_9 , all pass through this line, and hence form Neighbourhood Planes. (see Figure 12)

Since half-spaces H_1 and H_3 are defining the edge, we initialize the circular order as $(H_1^+, H_3^-, H_1^-, H_3^+)$, indicating that we are traversing around the edge circularly by entering half-space H_1 (denoted by +) leaving (denoted by -) half-space H_3 and so on. As we find half-spaces H_7 and H_9 to contain this edge, we

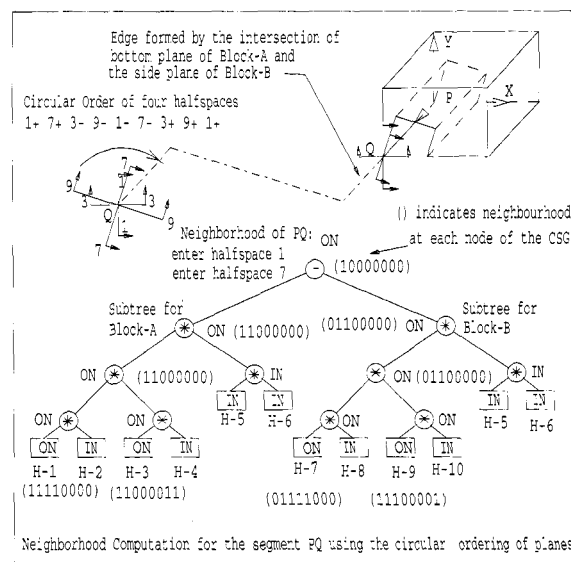


Figure 12: Edge neighbourhood computation: Four half-spaces contain the edge PQ formed by bottom plane of Block A and side plane of block A. The CSG tree and the neighbourhood vector at each node is shown.

insert them in the circularly ordered list by using the “*is symmetric test*” and the final circular ordering becomes $(H_1^+, H_7^+, H_3^-, H_9^-, H_1^-, H_7^-, H_3^+, H_9^+)$. Since there are eight distinct intervals in this circular list, the segment neighbourhood vector is initialized to eight bits.

For a leaf which is classified as OUT in the CSG tree all bits of this vector are initialized to zero, for a leaf which is classified as IN all bits are initialized to one. For a leaf which is classified as ON, the half-space this leaf is pointing to is then used to mark the bits of the segment neighbourhood vector. The sector, where we enter the half-space is marked one, all remaining sectors until we leave the half-space are also marked one. The rest of the sectors are empty and hence they are marked zero. For

example, for the leaf which is pointing to half-space H_3 the neighbourhood vector will be “11000011”.

Once we have done the initialization for the leaf nodes, we compute the neighbourhood vector of the internal nodes based on the Boolean operator at the node and the neighbourhood vectors of the left and right child, on a bit by bit basis.

For retained edge segments, the segment neighbourhood vector at the root of the CSG tree will contain ones and zeroes. This information is used along with the circularly ordered list to define the edge face adjacency relation. By using this adjacency relation and the vertex star neighbourhood of vertices, edges and faces one can construct the SGC of the CSG composition.