Geometric Classification Tests Using Interval Arithmetic in B-rep Solid Modeling

M. de S. G. Tsuzuki and M. Shimada
Escola Politécnica Universidade de São Paulo
Departamento de Engenharia Mecâtronica e Sistemas Mecânicos
Av. Prof. Mello Moraes, 2231
05508-900 São Paulo, SP. Brazil
mtsuzuki@usp.br

In this work, the use of interval arithmetic is considered to increase robustness of geometric classification algorithms in B-Rep solid modeling systems. The classification algorithms, also known as incidence tests, are important to keep the consistency between topology and geometry in a solid model during the application of Boolean operations. An error in the incidence test has deep impact over the steps that follow the Boolean operations and can damage the result, generating an inconsistent solid. The interval arithmetic incorporates approximation errors, so that, it eliminates the need of defining a fixed tolerance to do the comparison between floating-point numbers. However, it is not possible to directly convert floating-point algorithms to interval arithmetic, so that, it is necessary to reformulate the entire algorithm. Another important step in the Boolean operation is the determination of intersection points where the use of interval arithmetic can have side effects as intervals with large dimensions, and may cause incidence tests failures. It is necessary to control the growth of the intervals based on the geometry and topology. This work will introduce the application of interval arithmetic to a B-Rep solid modeler.

Keywords: *Solid modeling, interval arithmetic, geometric algorithms*

Introduction

The expansion of the application domain of geometric modeling in more and more engineering fields is making the robust implementations of the underlying geometric computations very essential. Otherwise the predictable characteristics of the design methodology can not be completely guaranteed. In a geometric modeling system, two types of information describe solids: the geometric and topological information which are related to the spatial relationships among geometric elements (face, edge, vertex, etc.). And the numerical information which describes the exact location of each entity in the three dimensional space. In continuous geometry these two types of information complement each other. The geometric and topological information can either be determined or verified by the numerical information and vice-versa. Usually all geometric algorithms are designed and implemented with this assumption of continuity. In practice, discreteness prevails at every stage in a computer-integrated design and manufacturing system. In the input stage the data is chosen from a discrete domain either by a CCD camera or a graphics user interface raster screen. In the computation stage all the numerical results are computed with the bounded precision of the CPU. Finally, at the output stage, the result is either displayed by the same graphics user interface raster, or translated into action by some actuator or a CNC machine whose movement is controlled by a discrete stepper motor. In this scenario the algorithms that do not take into account the discreteness of the system often fail with severe consequences.

Good software development uses fundamental strategies of layering and folding. The purpose of layering is to manage complexity and to achieve reuse of code components. The purpose of folding special cases is to achieve more compact code and to reduce thereby the opportunity for errors. These activities are so fundamental that most developers no longer think about them and consciously consider more the derived aspects such as complexities of code interdependence etc., as described by Lakos (1996). In the development of geometric software, however, we are required to review the code development fundamentals as part of the algorithmic strategy, and must re-examine assumptions that have become automatic.

Paper accepted October, 2003. Technical Editor: Edgar Nobuo Mamiya.

Geometric algorithms, though probably correct in a formal sense, often fail when implemented on a computer. The failure occurs as a result of the limited precision that is inherent to the interval representation on floating point numbers. One must always consider that any sequence of operations on a digital computer is essentially equivalent to a finite sequence of manipulations on a discrete grid of points. The absence of perfect arithmetic leads the program to make guesses where it should make accurate decisions, and often leads to failure.

Consider the example shown in Figure 1. It is determining the intersection between a line and a solid. In this example, assume that the front face of the cube cuts the line at a shallow angle. When carried out numerically, it is possible that we calculate the line intersection with the front face to be in the face interior, owing to the steep angle and the resulting better numerical conditioning of the linear system that determines the coordinates of the intersection. Nevertheless, the intersection with the top face may be calculated to lie on the edge of the face. After all, the shallow angle of the line with the top face results in a linear system that has a poorer condition number and therefore yields coordinate values with greater errors. The partition of the top and front faces that would result from this incidence is show in Figure 1. The two partitions are logically inconsistent, because the line now intersects the top face twice, at A and B. This will break the algorithms, as the designer would not anticipate that a line intersects a plane in two separated points.

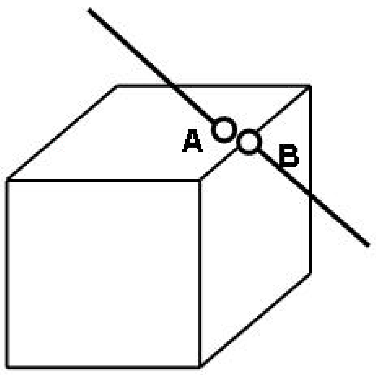

Figure 1. An edge intercepts a solid inconsistently.

The B-Rep Solid Modeling

The B-Rep representation stores several details of the solid such as how faces, edges and vertices are agglutinated to form a solid. A solid modeled by the B-Rep representation must have the capacity to describe how each face is connected to its adjacent faces, such that a closed volume is totally defined. The adjacency of those faces can be derived through some numerical techniques; however, the computational cost is very high and numerical precision problems can arise. In the B-Rep representation that information is explicitly available.

The first B-Rep based Solid Modeler was implemented using the Winged Edge Data Structure (Baumgart, 1975). However, during the past twenty years several data structures have been proposed to implement the B-Rep representation. The proposed data structures have in common the fact that they are based on the edge. In a curved environment where it is possible to an edge to be adjacent to two identical faces or to two identical vertices, the access to the topological information must be made with special care. The Winged-Edge Data Structure evoluted to an easier mechanism to access the topological information in a curved environment, called as Half-Edge Data Structure (Weiler, 1985).

A typical B-Rep data structure is shown in Figure 2. The solid is a collection of shells. The shell is a collection of adjacent faces. A face has one external boundary and zero or more internal boundaries. The loop represents the boundary of a face and the sequential list of half-edges that defines that boundary. An edge is associated to two half-edges, each half-edge is associated to one of the two faces and one of the two vertices associated to that edge.

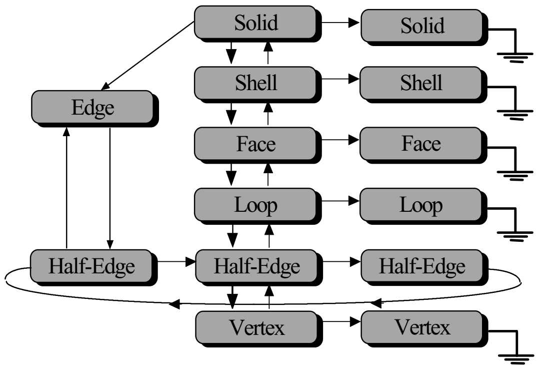

Figure 2. **Winged Edge Data Structure.**

Why do Geometric Algorithms Fail?

The first cause of an ambiguous branch is roundoff error. A floating-point number is represented with two components: a mantissa and an exponent. If two numbers, a and b, are represented with P-bit mantissas, then the product, ab, must in general be represented with a 2P-bit mantissa. However, to keep the size of these numbers from growing, floating-point arithmetic rounds the mantissa of ab to P bits. Thus, if a third number c is subtracted from ab the sign of $ab - c$ is not known when $|ab - c| < 2^{-P}|ab|$.

For that same reason $(1.0/10.0)*10.0$ on most systems does not compare equal to 1.0. That is because of rounding in the division. Since one tenth cannot be represented exactly as a binary fraction (it has an infinite expansion, just like one third as a decimal), some bits get lost. Multiplying by ten cannot restore those lost bits, so the result is not exactly one. For most purposes, though, it is close enough. Most algorithms attempt to the notion of close enough is to define an acceptable tolerance and check whether the difference between the two values being compared is less than this tolerance:

$$|a - b| < 0.0001 \Rightarrow a = b \tag{1}$$

However, if a and b are numbers with large exponents, 0.0001 is too small to be meaningful. The difference between $1.0001e40$ and $1.0000e40$ is $0.0001e40$, which is much larger than 0.0001. For large numbers, the only difference that is smaller then 0.0001 is zero. The test, also fails for small numbers, but in the opposite direction: the difference between 0.0001 and 0.00019 is less than 0.0001, but these two numbers differ by nearly a factor of two. They are close enough under this simple test, but often this is not we want. A solution is to scale the tolerance according to the magnitude of the numbers that we are dealing with. A simple approach is to divide the difference by one of the numbers:

$$\left| {(a - b)}/{b} \right| < 0.0001 \Rightarrow a = b \tag{2}$$

A central difficult for the reasoning approach is that deductions about geometric incidences are based on a notion of nearness. As such, they must follow unfamiliar logical rules. For example, fix a threshold distance below which is judged that the points are coincident. One may find that point A is coincident with point B and point B coincident with point C, but that point A is not coincident with point C.

The second cause of ambiguous branch is inaccurate input. For example, suppose a polyhedron is represented with a set of plane equations (one for each face) and some topological information. It is possible that the topological information specifies that four faces are to meet at a vertex, but the actual faces (lying on planes having floating point coefficients) do not meet at a unique vertex.

An important component of a solid modeler system is the Boolean operation engine. The Boolean operation engine aids the creation of complex solids as a combination of simple solids. One of the critical points in the Boolean operations engine is the classification algorithm among geometric element (vertex, edge and face) (Hoffman, 1989). It is necessary to determine if a solid's vertex is positioned on another solid's vertex, or if a solid's edge intersects another solid's edge defining an intersection vertex. The classification routines make verifications to determine if a geometric element is incident to another geometric element. The incidence tests studied in this work are the following:

- Incidence test between two vertexes (it is determined if two vertexes have equal coordinates);
- Incidence test between vertex and edge (it is determined if the vertex is on the edge); and
- Incidence test between vertex and plane (it is determined if the vertex is on the plane).

It is usual to implement algorithms that, using numeric manipulation with fixed tolerance, determine the geometric classification.

Solution Approaches

Several authors (Fortune, 1997) use exact arithmetic to address the robustness problem: this could be integer arithmetic, extended precision arithmetic or symbolic arithmetic. There are two problems that complicate this approach: proliferation, if the input to a geometric operation has k-digit precision, the output may require higher precision and irrationality, some operations result in coordinates that have no finite precision. In the literature (Latham, 1996), it is possible to find the symbolic reasoning approach: based on the nature of the geometric problem, one could symbolically reason, based on deductions have already been made, without any calculations, and deduce results that are a consequence of already known facts. A key issue in geometric computations is to achieve consistent evaluation of predicates and constructors. Therefore the problem of recognizing that such a decision is implied by earlier decisions already made lied in the domain of geometric reasoning.

The third is reliable calculations: using interval arithmetic, one can probably enclose the result of an arithmetic calculation with a floating point interval within which the result of the corresponding infinite precision exact computation must lie. In this work it is considered the use of interval arithmetic (Hu et al., 1996) to increase the robustness of the geometric classification algorithms. The interval arithmetic incorporates the approximation errors, and then it eliminates the need to define a tolerance to implement the comparison between two real numbers. Several authors (Hu et. al., 1996) studied the use of interval arithmetic in Solid Modeling. However, the implications of using interval arithmetic in the geometric classification were not discussed in the literature. In this work, it is shown that the adaptation of a conventional floating-point algorithm into a new interval algorithm is not just a type substitution. Another drawback is that the interval can grow too much, becoming useless.

In the following sections, it is presented an introduction to interval arithmetic, the representation of geometric elements using interval arithmetic and the incidence tests using interval arithmetic are discussed. Finally, some results and some conclusions are presented.

Rounded Interval Arithmetic

An interval of real numbers is defined by:

$$[a,b] = \{x \mid a \le x \le b\} \tag{3}$$

The interval $[a,b]$ is degenerated if $a = b$. The intersection between two intervals $[a,b]$ and $[c,d]$ is empty if $a > d$ or $c > b$. Otherwise:

$$[a,b] \cap [c,d] = [\max(a,c), \min(b,d)] \tag{4}$$

The union of two intervals that have a non-empty intersection is:

$$[a,b] \cup [c,d] = [\min(a,c), \max(b,d)] \tag{5}$$

The comparison between two intervals can result in three possibilities: certainly equal, possibly equal or certainly not equal. Two intervals $[a,b]$ and $[c,d]$ are considered certainly equal if $a = c$ and $b = d$. Two intervals $[a,b]$ and $[c,d]$ are considered certainly not equal if there is an empty intersection between them. Two intervals $[a,b]$ and $[c,d]$ are considered possibly equal if there is a non-empty intersection between them.

If floating-point arithmetic is used to evaluate the interval arithmetic equations there is no guarantee that the rounding of the bounds are performed conservatively. Then, rounded interval arithmetic ensures that the computed end points always contain the exact interval as follow (Abrams et. al., 1998):

$$\begin{aligned}
[a,b]+[c,d] &= [a+c-\varepsilon_l, b+d+\varepsilon_u] \\
[a,b]-[c,d] &= [a-d-\varepsilon_l, b-c+\varepsilon_u] \\
[a,b]\cdot[c,d] &= \begin{bmatrix} \min(a\cdot c, a\cdot d, b\cdot c, b\cdot d)-\varepsilon_l, \\ \max(a\cdot c, a\cdot d, b\cdot c, b\cdot d)+\varepsilon_u \end{bmatrix} \\
[a,b]/[c,d] &= \begin{bmatrix} \min(a/c, a/d, b/c, b/d)-\varepsilon_l, \\ \max(a/c, a/d, b/c, b/d)+\varepsilon_u \end{bmatrix}
\end{aligned} \tag{6}$$

Where ε_l and ε_u are the units in last place for each separate floating point number resulting from the floating point operations giving the lower and upper bonds. When performing standard operations, the lower bound is extended to include its previous consecutive floating-point number, which is smaller than the lower bound by ε_l. Similarly, the upper bound is extended by ε_u to include the next consecutive number. Thus the width of the result is enlarged by $\varepsilon_l + \varepsilon_u$ and the result will be reliable in subsequent operations.

The following example shows that equivalent algebraic expressions can give different results in the interval arithmetic. Consider the expression below:

$$f(x) = \frac{x}{1-x} \quad x \ne 1 \tag{7}$$

And $[x] = [2,3]$, then

$$f([x]) = \frac{[x]}{1-[x]} = \frac{[2,3]}{1-[2,3]} = [-3,-1] \tag{8}$$

For $x \ne 0$ we can rewrite $f(x)$ as

$$f_1(x) = \frac{x}{1-x} = \frac{1}{1/x - 1} \quad x \ne 0 \tag{9}$$

For the interval arithmetic evaluation over $[2,3]$ one obtains:

$$f_1([x]) = \frac{1}{\frac{1}{[x]}-1} = \frac{1}{\frac{1}{[2,3]}-1} = [-2, -3/2] \ne [-3,-1] \tag{10}$$

The reason for this is based on the fact that interval arithmetic does not follow the same rules as the arithmetic for real numbers.

Geometric Elements

The representation of the geometric elements is presented: vertex, line, vector and plane.

Interval Vertex: each coordinate of the vertex is represented by an interval. A 2D interval point is represented by $([x_{al}, x_{au}], [y_{al}, y_{au}])$, and a 3D interval point by $([x_{al}, x_{au}], [y_{al}, y_{au}], [z_{al}, z_{au}])$.

According to Hu and Patrikalakis (1996), the incidence transitivity is an example of the superior robustness of interval arithmetic against floating-point arithmetic. Using floating-point implementation, a point A can be incident to point B, and point B can be incident to point C. But it is not sure that point C is incident to point A, as illustrated in Figure 3 (a). Using interval arithmetic points A and B are replaced by a new point that covers both points. So, point C is incident to this new point. In this case the interval grows (see Figure 3 (b)).

The widths of intervals of interval points are typically very small. However, to make the transitivity incidence property true, the widths of intervals of interval points can increase very fast.

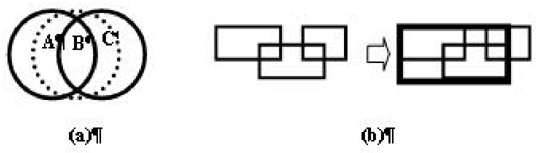

Figure 3. Incidence transitivity using tolerance is not true, and using interval arithmetic it becomes true.

Interval Line: Intervals line, with two end vertexes $[\mathbf{P}_0]$ and $[\mathbf{P}_1]$, are defined as the linear combination of these two interval vertices. The 2D representation of the interval line is a polygon with six sides, and the 3D representation is a polyhedron.

Interval Vector: a vector is defined by:

$$[\mathbf{v}] = \overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]} = [\mathbf{P}_2] - [\mathbf{P}_1] \tag{11}$$

It is shown in Figure 4, an example where $[\mathbf{P}_1] = ([0,1],[0,1])$ and $[\mathbf{P}_2] = ([2,3],[2,3])$. The region defined by vector $[\mathbf{v}] = ([1,3],[1,3])$ contains all possible floating point vectors between $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$ (see Figure 5). It will be shown in a next section that in the discussion of Vertex-Edge incidence, the direction of the vector is more important than the value of its module. As the size of the interval vertexes grows and the distance between them decreases, the number of possible directions that can be associated to the interval vector increases.

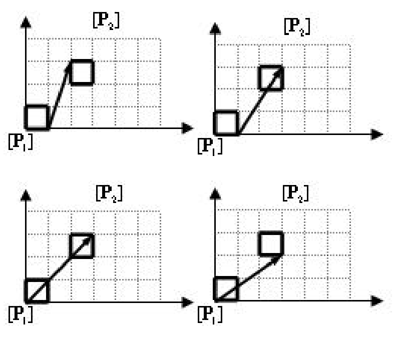

Figure 4. An Interval Vector between vertexes $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$.

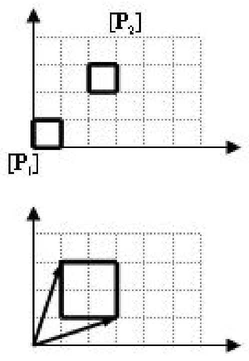

Figure 5. Representation of interval vector $\overrightarrow{[\mathbf{P}_1\mathbf{P}_2]}$.

Another very important property is that the size of the interval always increases. Consider the calculation of $[\mathbf{P}_{2n}] = [\mathbf{P}_1] + \overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]}$, the result is $[\mathbf{P}_{2n}] = ([1,4],[1,4])$. Actually, this result should be equal to point $[\mathbf{P}_2]$ (see Figure 6). However, $[\mathbf{P}_{2n}]$ is a larger region than $[\mathbf{P}_2]$. As we have seen previously, the evaluation in interval arithmetic of mathematically equivalent expressions gives values that are not necessarily equal.

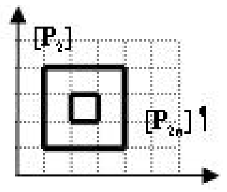

Figure 6. Example showing that $[\mathbf{P}_2]$ is not the same as $[\mathbf{P}_{2n}]$.

Interval Face: the face equations are calculated based on vertexes coordinates. An efficient and numerically robust way to calculate the face equation is the method of Newell (Mäntylä, 1988).

Considering that the face is defined by n vertexes, and the vertexes' coordinates are $[x_i]$, $[y_i]$ and $[z_i]$. Then, the coefficients $[a]$, $[b]$, $[c]$ and $[d]$ of the plane equation can be calculated by:

$$\begin{aligned} [a] &= \sum_{i=1}^{n} ([y_i]-[y_{i+1}])([z_i]+[z_{i+1}]) \\ [b] &= \sum_{i=1}^{n} ([z_i]-[z_{i+1}])([x_i]+[x_{i+1}]) \\ [c] &= \sum_{i=1}^{n} ([x_i]-[x_{i+1}])([y_i]+[y_{i+1}]) \end{aligned} \tag{12}$$

The remaining coefficient is calculated using the average point in the following equation:

$$[d] = -\left[[x_{av}] \quad [y_{av}] \quad [z_{av}]\right] \cdot \left[[a] \quad [b] \quad [c]\right]^T \tag{13}$$

Incidence Tests

In this section, we will analyze three incidence tests: vertex-vertex, vertex-edge and vertex-face.

Vextex-Vertex: Given two vertexes $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$, where:

$$\begin{aligned} [\mathbf{P}_1] &= ([x_{1l}, x_{1u}],[y_{1l}, y_{1u}],[z_{1l}, z_{1u}]) \\ [\mathbf{P}_2] &= ([x_{2l}, x_{2u}],[y_{2l}, y_{2u}],[z_{2l}, z_{2u}]) \end{aligned} \tag{14}$$

Vertex $[\mathbf{P}_1]$ is considered incident to vertex $[\mathbf{P}_2]$ if the following conditions are satisfied:

$$\begin{aligned} [x_{1l}, x_{1u}] \cap [x_{2l}, x_{2u}] &\neq \varnothing \\ [y_{1l}, y_{1u}] \cap [y_{2l}, y_{2u}] &\neq \varnothing \\ [z_{1l}, z_{1u}] \cap [z_{2l}, z_{2u}] &\neq \varnothing \end{aligned} \tag{15}$$

If $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$ are incidents, then a new vertex $[\mathbf{P}_3]$ is created. Vertex $[\mathbf{P}_3]$ covers vertexes $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$.

Mäntylä (1988) and Chiyokura(1988) used fixed tolerance to determine if two vertexes are coincident. Guibas et al. (1989) determines a pair $(e.lo, e.hi)$ such that $e.lo \leq 0.5\|p,q\| \leq e.hi$. Where

$\|p,q\| = \sqrt{(p.x-q.x)^2+(p.y-q.y)^2}$. The interval $(e.lo, e.hi)$ shows that it is possible to make p and q coincident if one displaces both points by $e.hi$ in suitable direction, and it is not possible to make p and q coincident if one displaces them by less than $e.lo$. The tolerance used is calculated, based on the floating point values involved in the calculation. However, none of the proposals deals with the transitivity property.

Vertex-Edge: The tests for collinearity and orientation of three given points deserve careful discussion, since they are basic building blocks of many geometric algorithms. Given three vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$. One method to verify if vertex $[\mathbf{P}_3]$ is on $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$ is to calculate the area defined by the triangle formed by vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$. If the area defined by triangle $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$ is null, then the three vertexes are colinear, and there is a possibility that vertex $[\mathbf{P}_3]$ is on $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$. The area is calculated by:

$$\overrightarrow{[\mathbf{A}]} = (\overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]} \times \overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]})/2 \tag{16}$$

Then a scalar product is calculated to determine if vertex $[\mathbf{P}_3]$ is between $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$ or not:

$$[E] = \overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]} \cdot \overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]} \tag{17}$$

If the scalar product is less than or equal to zero, then vertex $[\mathbf{P}_3]$ is considered incident to $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$. When working with vertexes where the distance between them is almost the size of the interval some unexpected results can appear, as the number of directions associated to the interval vector can increase. The area is determined considering $[\mathbf{P}_1]$ as the common vertex and two vectors emanate from it. When using interval arithmetic, depending on the common vertex, the distance among the points and the size of the intervals the result can change. Consider the example shown in Figure 7, where $[\mathbf{P}_3]$ is not on edge $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$ and $[\mathbf{P}_1] = ([0,2],[0,2])$, $[\mathbf{P}_2] = ([4,6],[6,12])$ and $[\mathbf{P}_3] = ([4,4],[2,2])$. In case that $[\mathbf{P}_1]$ is the common vertex, the calculation results in:

$$\overrightarrow{[\mathbf{A}_1]} = (\overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]} \times \overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]})/2 = [-24,2] \tag{18}$$

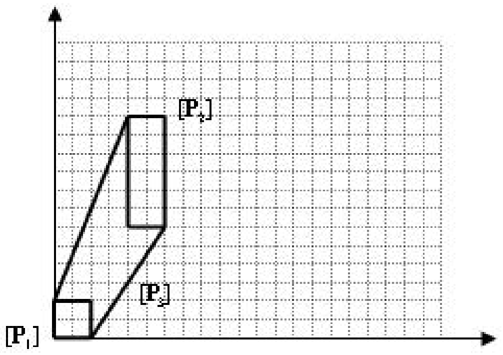

Figure 7. The distance among vertexes is near the interval vector's size.

However, when $[\mathbf{P}_3]$ is the common vertex, the calculation results in:

$$\overrightarrow{[\mathbf{A}_2]} = (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]} \times \overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]})/2 = [-20,-2] \tag{19}$$

$[\mathbf{A}_1]$ is probably equal zero giving that $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$ are probably collinear. However, $[\mathbf{A}_2]$ is certainly not equal zero, concluding that $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$ are certainly not collinear. Then, to be sure of the result, vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$ are certainly collinear when $[\mathbf{A}_1]$, $[\mathbf{A}_2]$ and $[\mathbf{A}_3]$ are possibly equal to zero, where:

$$\begin{aligned}\overrightarrow{[\mathbf{A}_1]} &= (\overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]} \times \overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]})/2\\ \overrightarrow{[\mathbf{A}_2]} &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]} \times \overrightarrow{[\mathbf{P}_3][\mathbf{P}_2]})/2\\ \overrightarrow{[\mathbf{A}_3]} &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]} \times \overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]})/2\end{aligned} \tag{20}$$

To determine if vertex $[\mathbf{P}_3]$ is on $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$, the following three scalar products must be calculated:

$$\begin{aligned}[E_1] &= \overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]} \cdot \overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]}\\ [E_2] &= \overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]} \cdot \overrightarrow{[\mathbf{P}_3][\mathbf{P}_2]}\\ [E_3] &= \overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]} \cdot \overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]}\end{aligned} \tag{21}$$

If $[E_3]$ is probably less than or equal to zero, then this means that vertex $[\mathbf{P}_3]$ is probably on $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$. If $[E_1]$ and $[E_2]$ are probably greater than zero, then this means that $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$ are respectively probably outside $\overline{[\mathbf{P}_2][\mathbf{P}_3]}$ and $\overline{[\mathbf{P}_1][\mathbf{P}_3]}$ (see Figure 8). If any of these verifications fails, then point $[\mathbf{P}_3]$ is certainly not on $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$.

This way, it is necessary to make three cross products and three scalar products. Trying to find new possibilities to verify the geometric incidence, it was verified that if the order of the calculations is inverted the number of calculations could be smaller. First it is verified the three scalar products to determine the relative position between $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$, and it is necessary only one cross product to verify if $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$ are collinear in the case that $[\mathbf{P}_3]$ is between $[\mathbf{P}_1]$ and $[\mathbf{P}_2]$:

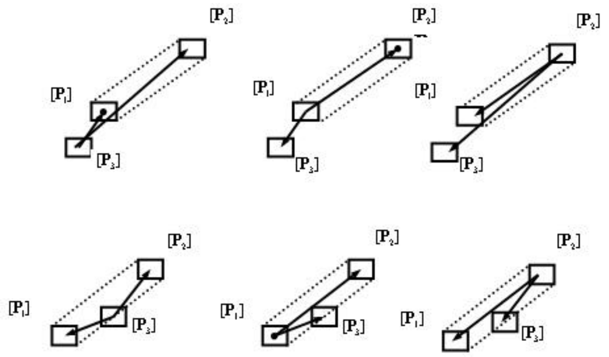

Figure 8. Possible configurations between vertex $[\mathbf{P}_3]$ and edge $\overline{[\mathbf{P}_1][\mathbf{P}_2]}$.

$$\begin{aligned}[E_1] &= \overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]}\cdot\overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]}\\ [E_2] &= \overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]}\cdot\overrightarrow{[\mathbf{P}_3][\mathbf{P}_2]}\\ [E_3] &= \overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]}\cdot\overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]}\\ [\mathbf{A}] &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]}\times\overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]})/2\end{aligned} \tag{22}$$

Mäntylä (1988) and Chiyokura (1988) used fixed tolerance to determine the incidence between edge and vertex. Guibas et al. (1989) determine the smallest perturbation that makes the three points collinear. However, it is not possible to differentiate inputted from calculated vertexes' coordinates. In this approach, inputted vertex' coordinates have intervals with smaller size than calculated vertex' coordinates. Hu et. al. (1996) do not analyze this incidence test.

Vertex-Face: It is verified if vertex $[\mathbf{P}_4]$ is on the plane defined by vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$. One method is to calculate the volume of the tetrahedron defined by vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$, $[\mathbf{P}_3]$ and $[\mathbf{P}_4]$. If the volume $[V]$ is equal to zero, then $[\mathbf{P}_4]$ is on the plane, where $[V]$ is given by:

$$[V] = (\overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]}\times\overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]})\cdot\overrightarrow{[\mathbf{P}_4][\mathbf{P}_1]}/6 \tag{23}$$

Similarly to the Vertex-Edge incidence, using interval arithmetic, it is necessary to calculate four volumes. Otherwise, an incorrect answer can be found. When $[V_1]$, $[V_2]$, $[V_3]$ and $[V_4]$ are possibly equal to zero then $[\mathbf{P}_4]$ is on the plane defined by vertexes $[\mathbf{P}_1]$, $[\mathbf{P}_2]$ and $[\mathbf{P}_3]$. Where:

$$\begin{aligned}[V_1] &= (\overrightarrow{[\mathbf{P}_2][\mathbf{P}_1]}\times\overrightarrow{[\mathbf{P}_3][\mathbf{P}_1]})\cdot\overrightarrow{[\mathbf{P}_4][\mathbf{P}_1]}/6\\ [V_2] &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_2]}\times\overrightarrow{[\mathbf{P}_3][\mathbf{P}_2]})\cdot\overrightarrow{[\mathbf{P}_4][\mathbf{P}_2]}/6\\ [V_3] &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_3]}\times\overrightarrow{[\mathbf{P}_2][\mathbf{P}_3]})\cdot\overrightarrow{[\mathbf{P}_4][\mathbf{P}_3]}/6\\ [V_4] &= (\overrightarrow{[\mathbf{P}_1][\mathbf{P}_4]}\times\overrightarrow{[\mathbf{P}_2][\mathbf{P}_4]})\cdot\overrightarrow{[\mathbf{P}_3][\mathbf{P}_4]}/6\end{aligned} \tag{24}$$

Another method consists in use the plane's equation $a\cdot x+b\cdot y+c\cdot z+d=0$ and applies it to vertex point $[\mathbf{P}_4]$. If $a\cdot[x_4]+b\cdot[y_4]+c\cdot[z_4]+d=0$ is possibly equal to zero, then point $[\mathbf{P}_4]$ is on the plane. This is a much simpler approach and it is used in our solid modeler implementation.

Results – Boolean Operations

According to Mäntylä (1988), the lack of robustness to deal with all the possible intersections that can happen between the geometric elements and the possibility of precision error during the incidence tests and the determination of intersection are the main problems that the Boolean operations can suffer. For the second problem, the rounded interval arithmetic will help to keep the topologic consistency preventing that the approximation errors interfere in the solid topology. The algorithm of Boolean operation of type union can be divided into the following steps, for two solids $\mathbf{A}$ and $\mathbf{B}$:

1. Determination of the intersection points of faces from solid $\mathbf{A}$ with the faces from solid $\mathbf{B}$;
2. The intersection points determined in the previous step are transformed into vertexes in the data structure. Edges and faces are also added to the data structure;
3. The faces from solids $\mathbf{A}$ and $\mathbf{B}$ are classified to determine which face must be removed (Chiyokura, 1988);
4. After the deletion of the faces, adjusts and cleaning of the data structure of solids $\mathbf{A}$ and $\mathbf{B}$ are made. This is necessary because the region's topology where is the gluing is going to happen must match exactly at both solids;
5. Solid $\mathbf{B}$ is glued to solid $\mathbf{A}$. This means that the data structure of solid $\mathbf{B}$ is moved to the data structure of solid $\mathbf{A}$.

In steps 1 and 2, the interval arithmetic has fundamental importance, to prevent that unnecessary vertexes, edges or faces are created, because unnecessary elements can change the solid's topology.

Plane-Edge Intersection

When the intersection point between an edge and a face is determined, the size of the interval point becomes bigger than the original edge. Figure 9 illustrates the intersection point larger than the original edge. This is a big problem, because as the interval grows it becomes useless. Figure 10, in the right, shows a result of a boolean operation where the interval size of calculated vertex' coordinates became too large, and consequently the incidence tests fails producing an incorret result.

However, it is known that the equation of a plane is a secondary information, calculated based on vertexes' coordinates. Thus, the size of a interval vertex must be bounded by the interval edge and not by the interval face, as shown in Figure 11.

This way, a verification routine is executed to limit the size of the interval point calculated by plane-edge intersection. Figure 10 shows an example of the Boolean Operations implemented using the interval correction (left) and without using the interval correction (right).

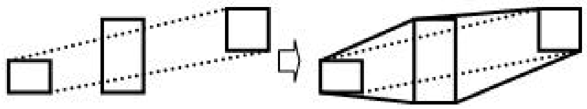

Figure 9. The intersection point is larger than the interval edge.

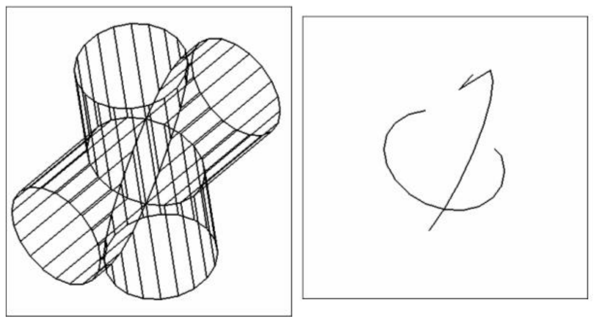

Figure 10. Example of a union of two cylinders with interval correction and without interval correction generating a true result and an incorrect result.

Fixed Tolerance x Interval Arithmetic applied to Boolean Operations

The following example compares the incidence tests using interval arithmetic and fixed tolerance. We have that $\mathbf{P}_1=(0,0)$, $\mathbf{P}_2=(-0.01,0)$ in the floating point environment. The fixed tolerance is equal to 0.001. In the interval arithmetic environment we have $[\mathbf{P}_1]=([0,0],[0,0])$ and $\mathbf{P}_2=([-0.01,0.01],[0,0])$.

The example in Figure 12 shows the intersection between solids $\mathbf{A}$ and $\mathbf{B}$. The result using fixed tolerance has a salient part, because of the difference between $\mathbf{P}_1$ and $\mathbf{P}_2$ is greater than the fixed tolerance. However, the result using interval arithmetic does not have this salience because the interval size made the calculations more robust and accumulated the rouding in the interval.

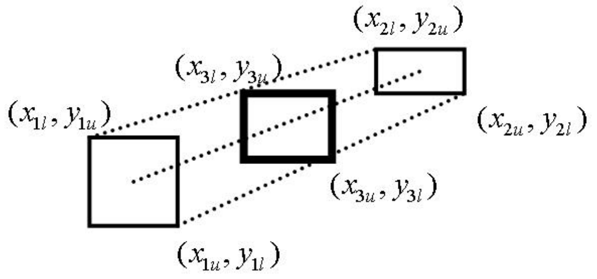

Figure 11. The intersection point was corrected using the border of the two interval vertexes.

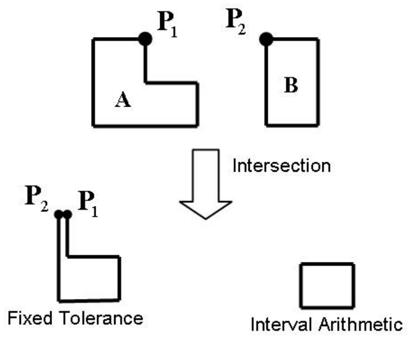

Figure 12. Comparison between results using floating point tolerance and interval arithmetic.

Conclusions

In this work it was applied interval arithmetic into a B-Rep solid modeler. It was demonstrated that the incorporation of interval arithmetic is not only a type substitution, and the algorithm must be totally reformulated. It was proposed a new algorithm to classify the incidence between vertex and edge, and applied the algorithm in the implementation of Boolean Operations. It was proposed a method to verify the size of the interval vertex calculated from plane-edge intersection, controlling its size.

The interval arithmetic can give more robust results than the commonly used floating point arithmetic. In a floating point arithmetic environment, it is necessary to manage the transitivity property in a higher level. If $\mathbf{P}_1$ is equal to $\mathbf{P}_2$ and $\mathbf{P}_2$ is equal to $\mathbf{P}_3$, however $\mathbf{P}_1$ is not equal to $\mathbf{P}_3$. All results obtained from the floating point arithmetic rules application are processed and $\mathbf{P}_1$ turns to be equal to $\mathbf{P}_1$ or $\mathbf{P}_2$ turns to be not equal to $\mathbf{P}_3$. In a interval arithmetic environment, this rule management is not necessary as the transitivity property is implemented in a very low level.

Acknowledgement

This work was partially supported by FAPESP and CNPq under grant 300.224/96.

References

Abrams, S.L.; Cho, W.; Hu, C.Y.; Maekawa, T.; Patrikalakis, N.M.; Sherbrooke, E.C.; Ye, X.; 1998, "Efficient and Reliable Methods for Rounded Interval Arithmetic", Computer Aided Design, 30(8), 657-665.

Baumgart, B., 1975, "A Polyhedron Representation for Computer Vision". In National Computer Conference, pp. 589-596, AFIPS Conf. Proc.

Chiyokura, H., 1988, "Solid Modeling with DESIGNBASE: Theory and Implementation", Addison Wesley Publishing Company.

Fortune, S., 1997, "Polyhedral Modelling with Multiprecision Integer Arithmetic", Computer Aided Design, 29(2), 123-133.

Guibas, L., Salesin, D., Stolfi, J., 1989, "Epsilon Geometry: Building Robust Algorithms from Imprecise Computations". In Proceedings of the Fifth Annual Symposium on Computational Geometry, 208-217, ACM Press.

Hoffmann, C.M., 1989, "Geometric and Solid Modeling: An Introduction", Morgan & Kaufmann.

Hu, C.Y.; Patrikalakis, N.M., Ye, X., 1996, "Robust Interval Solid Modeling Part I: Representations", Computer Aided Design, 28(10), 807-817.

Hu, C.Y.; Patrikalakis, N.M., Ye, X., 1996, "Robust Interval Solid Modeling Part II: Boundary Evaluation", Computer Aided Design, 28(10), 819-830.

Lakos, J.; 1996, "Large Scale C++ Software Design", Addison-Wesley, Reading, MA.

Latham, R. S.; Middleditch, A. E.; 1996, "Connectivity Analysis: a Tool for Processing Geometric Constraints", Computer Aided Design, 28(11), pp, 917-928.

Mäntylä, M., 1988, "An Introduction to Solid Modeling", Computer Science Press.

Weiler, K., 1985, "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments", IEEE Computer Graphics & Applications, 5(1):21-39.