Boolean Set Operations with Solid Models

The Boolean engine of HP PE/SolidDesigner applies standard and nonstandard Boolean set operations to solid models to perform an impressive variety of machining operations. Parallel calculation boosts performance, especially with multiprocessor hardware.

by Peter H. Ernst

Machining operations like punch, bore, and others play an important role in the function set of contemporary CAD systems. In HP PE/SolidDesigner, the impressive variety of machining commands are driven by a single topology engine, often referred to as *the Boolean engine*.

It might seem that the algorithm used by the Boolean engine would be extremely complex and esoteric, and this is indeed true in some respects. The underlying principles, however, are simple.[1] Most of this article demonstrates this by taking a fairly intuitive look at the internal machinery. This will provide a road map for the second, more technical part of the article, in which some key algorithms are explained in greater depth. Finally, some unusual applications of the Boolean engine are briefly mentioned.

Different Flavors of Solids

Before exploring the internals of the Boolean engine, let's take a look at the objects that it works on. These objects are called solids, or simply bodies. Solids, in our terms, are mathematical boundary representation (B-Rep) models of geometric objects. Fig. 1 shows a B-Rep model of a cylinder.

Usually several categories of solids are distinguished based on their manifold characteristics. For our purposes we just need to know that *manifold* solids represent real objects and *nonmanifold* solids are impossible in some way. Manifold bodies are of general interest, since they can be manufactured. Fig. 2 shows a screwdriver representing the class of manufacturable bodies.

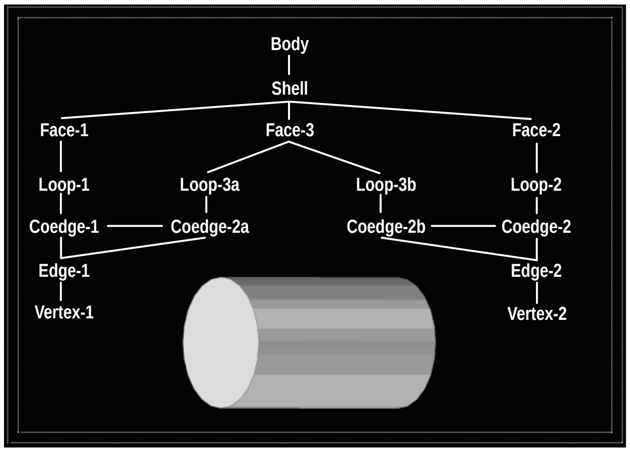

Fig. 1. A boundary representation (B-Rep) model of a cylinder.

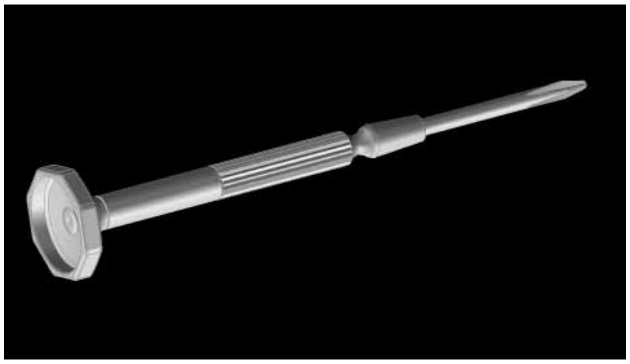

Fig. 2. A screwdriver representing the class of manufacturable bodies.

The class of nonmanifold bodies is the realm of the impossible bodies. These bodies cannot be manufactured because the material thickness goes to zero (that the thickness goes to zero is a consequence, not a cause of the nonmanifoldness). Nevertheless, they have have some importance as conceptual abstractions or simplifications of real (manifold) solids. Nonmanifold solids sometimes are (conveniently) generated as an intermediate step in the design process. They are also important to various simulation applications, and sometimes to finite-element analysis and NC machining. Fig. 3 shows a selection of nonmanifold bodies. To the left is

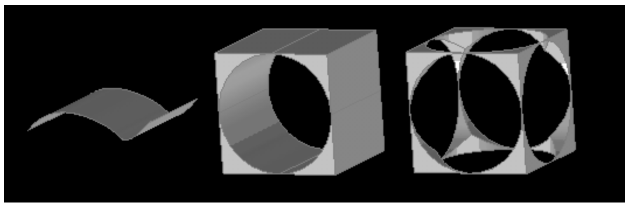

Fig. 3. Bodies that are nonmanufacturable because of (left) zero thickness in general, (center) zero thickness at edges, and (right) zero thickness at vertices.

a *sheet*, which has zero thickness in general. The middle solid is *edge nonmanifold*, having zero thickness at edges, and the right solid is *vertex nonmanifold*, having zero thickness at vertices.

The Boolean engine in its different guises is used to change bodies by the rules of Boolean set operations. In other words, it is able to combine two volumes using one of the three standard operators: subtract, unite, or intersect. The operation is performed on solids in the same way as on the sets of mathematical set theory. The effects of the standard operations on sets and volumes are illustrated in Fig. 4. The two bodies at the top of the picture are combined in three ways, using the three standard Boolean operations. The result of each Boolean operation is shown at the bottom.

An Intuitive Approach to the Boolean Engine

Now that we are equipped with the right background, we can explore the various stages of the Boolean algorithms. To do this we will use a thought experiment (such experiments are widely acknowledged as safe and cheap). To perform this experiment we only need some paint, a sharp knife, and some imagination.

Coloring. In the first stage both solids participating in the Boolean operation are filled with different colors, let's say yellow for one and blue for the other. Fig. 5 shows two bodies that have been set up for a Boolean operation and colored according to our rule. Let's assume that, unlike real solids, they can permeate each other without problems. Since the Boolean operation hasn't been performed yet the picture still shows two disjoint solids that just happen to overlap. To show what's going on inside the bodies, the yellow body has been made transparent.

Now we mark the lines where the two bodies permeate each other, let's say with red color. The red lines in Fig. 6 are called the *intersection graph*. The two solids are still disjoint.

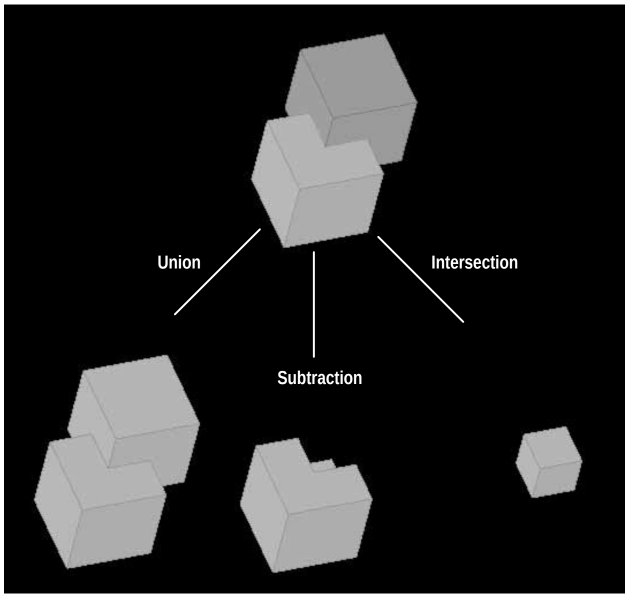

Fig. 4. Results of applying the standard Boolean set operations to two solid bodies.

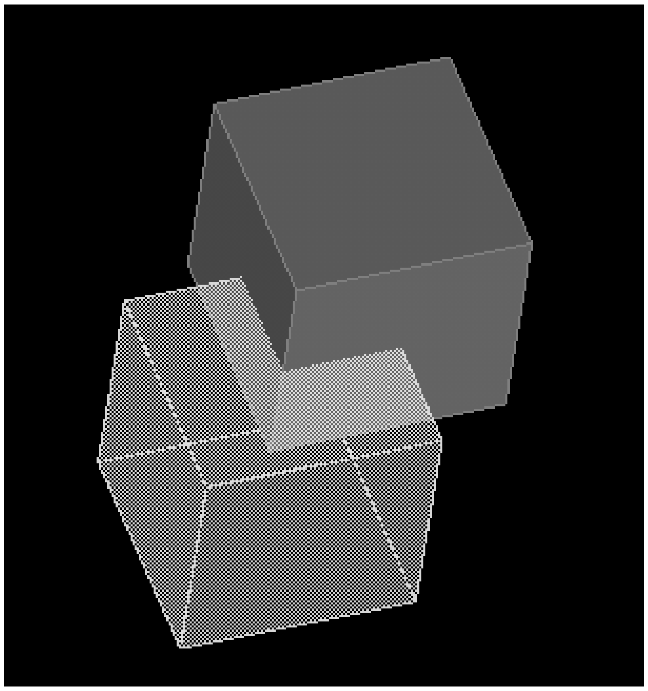

Fig. 5. Two disjoint solids that happen to overlap.

Making Soap Bubbles—Cellular Bodies. In the second stage we *knit* both solids together using the intersection graph. A structure very similar to those formed by soap bubbles is created, as shown in Fig. 7. The two solids now hang together at the intersection graph. In the space where both bodies overlap a green color can be seen. This is the mixture of yellow and blue. To get a better vision of the geometric situation some faces have been made transparent.

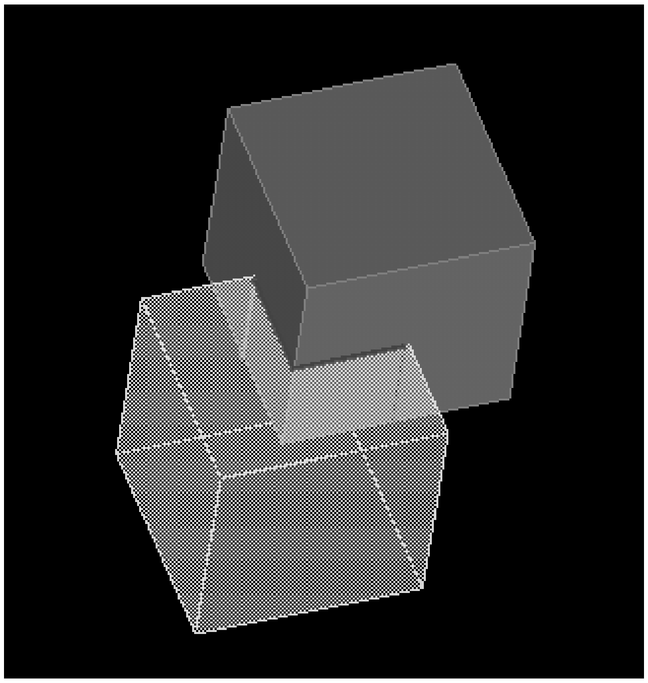

Fig. 6. Intersection graph (red).

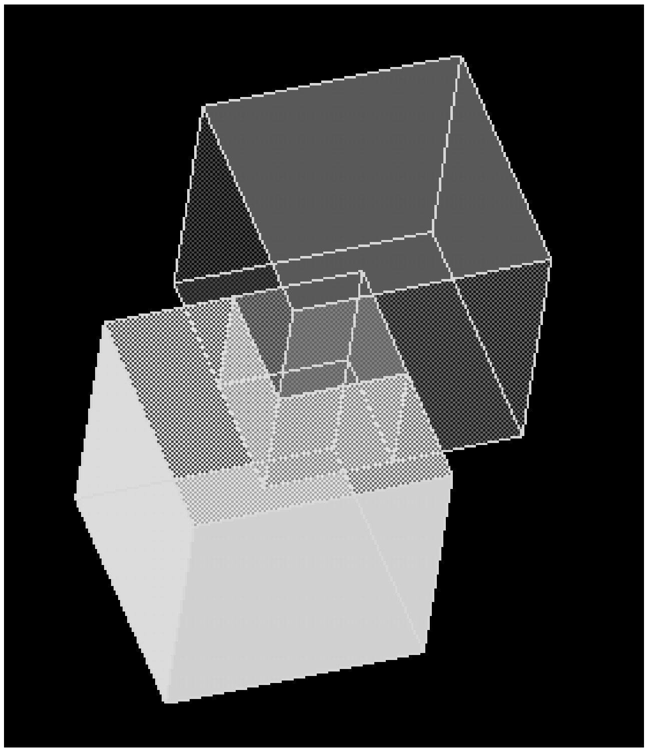

Fig. 7. Result of knitting the two bodies together at the intersection graph. Choosing a Boolean operation is now equivalent to deciding which colors to keep and which to delete.

Getting Rid of the Wrong Colors. In the third and last stage of our imaginary process not much is left to do. Up to now we have not said which kind of Boolean operation (union, subtraction, or intersection) we wanted. Now is the time to decide.

To get the desired result we simply pick the appropriate color and get rid of all volumes of a different color than the one we picked. Initially we chose two colors—blue and yellow—so we will find three colors in our soap bubble cluster: blue, yellow, and green. In regions where blue and yellow volumes overlap we get green. The table below shows which colors will be kept or deleted from the body depending on the particular type of Boolean operation we choose.

	Keep	**Delete**
Union	blue and yellow	green
Subtraction	yellow	green and blue
Intersection	green	blue and yellow

Easy, isn't it? Pat yourself on the back (and clean up the mess of paint and chipped-off pieces).

Technical Talk: The Boolean Algorithm

In the preceding example we only had to mark the lines where the color changes to obtain the intersection graph. The Boolean engine algorithm that does this is a bit more complex. To understand it we must again look at the mathematical representation of a solid. In Fig. 1 we have seen the general data structure layout of a cylinder. That sketch, however, lacks any explicit references to geometry. In HP PE/SolidDesigner's B-Rep structure, three base classes of geometries are used: points, curves, and surfaces. The last two have several subclasses. For example, a curve can be a straight line, circle, ellipse, or spline. In the following discussion the geometric subclasses are used for illustration purposes, but the Boolean algorithm itself does not depend on any specific geometry types, since it is implemented in a generic way.

Each geometry class has a corresponding topological carrier that puts it into perspective in the context of a solid model. The table below shows this relationship:

Topology		**Geometry**
Vertex	←	Point
Edge	←	Curve
Face	←	Surface

The topological entities face and edge are *smart* carriers because they not only *hold* their geometries, but also *bound* or *trim* them. To understand what this means we must realize that most geometries are of infinite extent, and even if they are finite only a small segment might be of interest.

Fig. 8 exemplifies the relationship between topology and geometry. Looking at the cylinder (sf3), notice that only a segment of the otherwise infinite cylindrical surface is used. This segment is called a face (fa3). Likewise, only two circular regions of the otherwise infinite planes sf1 and sf3 are used to close the cylinder. The circular regions are face fa1 and face fa2. (Note: The top and bottom faces of the cylinder have been lifted off a bit for better demonstration. The double yellow edges coincide in reality.)

The concept of trimmed surfaces is essential for the next section, because it introduces some unexpected complications when constructing the intersection graph.

Constructing the Intersection Graph. Earlier we simply used an excellent pattern recognizer called the human brain to find the lines where the color changes. Teaching this ability to a computer involves a considerable amount of mathematics.

Fig. 9 shows the construction of one segment of an intersection graph (a graph edge). The drawing shows two intersecting surfaces sf1 and sf2 carrying two faces fa1 and fa2. To construct the graph edge (the piece of the intersection track inside both faces) the following steps are required:

- The two unbounded surfaces sf1 and sf2 are intersected, giving the intersection track (track).

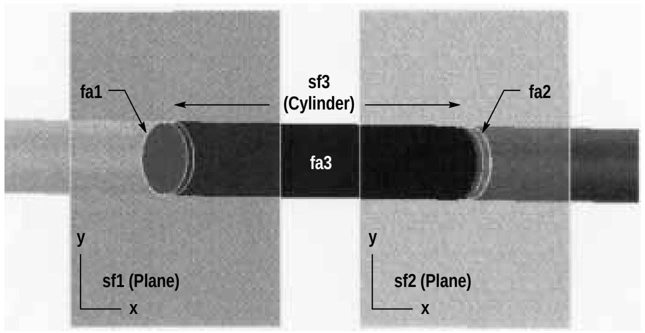

Fig. 8. An example of the relationship between topology and geometry. Faces and edges bound or trim their geometries, which consist of infinite curves and surfaces.

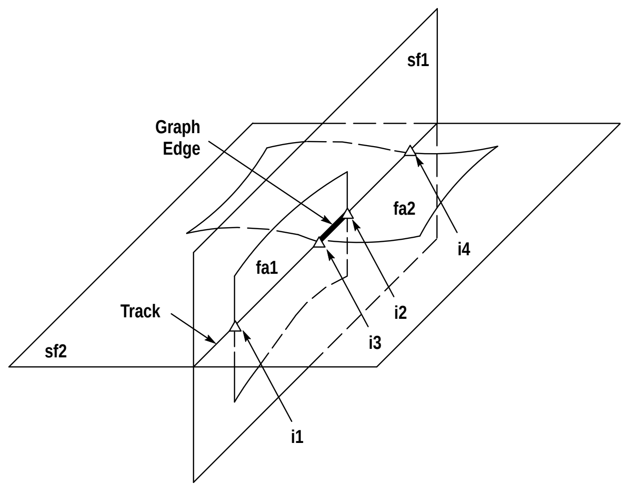

Fig. 9. Construction of one segment of an intersection graph (a graph edge).

- The edges of fa1 are intersected with surface sf2 to yield the edge/surface intersection points i1 and i2. Similarly, the edges of fa2 are intersected with sf1 giving the intersection points i3 and i4.
- The intersection points are ordered along the track.
- The ordered points are examined for their *approach values*. The approach values simply tell if a face is entered or left when passing a particular point. This information can be used to deduce the containment of a segment of the intersection graph with respect to its generating faces. The approach and containment values for the intersection points in the previous drawing are:

Point		**Containment with respect to:**	
	Approach	**fa1**	**fa2**
		outside	outside
i1	entering fa1		
		inside	outside
i2	entering fa2		
		inside	inside
i3	leaving fa1		
		outside	inside
i4	leaving fa2		
		outside	outside

- The segments of the intersection graph inside both faces are used to create the graph edge(s) of a particular intersection. In this example only the segment bounded by i2 and i3 fulfills this condition.

Parallelism. The complete intersection graph of two bodies is obtained by pairwise intersection of faces selected from both solids. The number of required face/face intersections depends on the number of faces in both solids:

$$i = nm,$$

where i is the number of intersections, n is the number of faces in one body, and m is the number of faces in the other body.

The number of required intersections grows rapidly (quadratically) with the complexity (number of faces) of the solids. Fortunately the different face/face intersections can be easily performed in parallel. The algorithm is structured such that it can create a cascade of *threads* (a sort of subprocess). For each pair of faces a subprocess is launched that splits itself to calculate the surface/surface intersections and the edge/surface intersections in parallel. With the availability of multiprocessor hardware the advantages of this algorithmic structure are seen as increased performance of the Boolean operations.

Imprinting and Coloring. In the intuitive approach, coloring the faces, that is, determining which pieces are inside or outside, was no problem because it could easily be seen. On the machine level other means are required.

Intersection tracks split surfaces and faces into left and right halves. Additionally, surfaces split space into halves called *half spaces*. We can classify each piece of the split face to a half space with respect to the other surface. This procedure is demonstrated in Fig. 10.

Classification is done with respect to the surface normals (colored arrows) of both surfaces (sf1 and sf2) and the intersection track.

Unusual Boolean Applications

It is easy to see that the Boolean engine is driving most machining operations. Here are some applications in which it is not so obvious.

Partial Booleans. Regular Boolean operations attempt to calculate all intersection tracks between bodies. In contrast, partial Boolean operations calculate only one intersection track. Which one depends on the particular application. One example of a partial Boolean operation in HP PE/SolidDesigner is wrapped into the extrude-to-part command. It fires a profile defined in a workplane onto a body as shown in Fig. 11. The picture shows a body and a profile set up for the

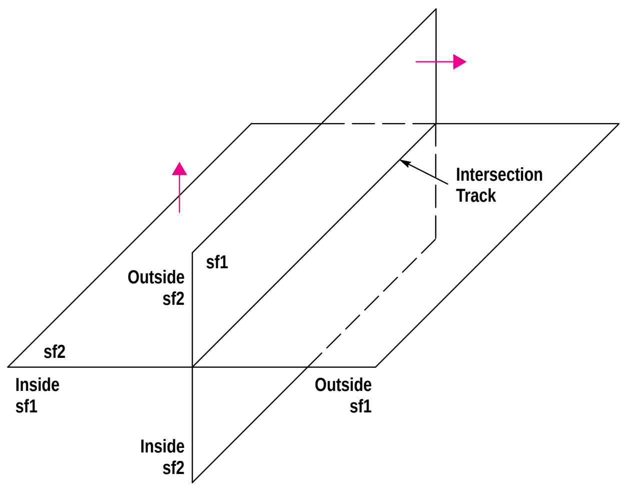

Fig. 10. Surfaces split space into halves called half spaces (inside and outside along surface normals). Each piece of a split surface can be classified as belonging to a half space with respect to the other surface.

Fighting Inaccuracies: Using Perturbation to Make Boolean Operations Robust

The robustness of Boolean operations between solids is crucial for the usability of a solid modeler like HP PE/SolidDesigner. Unfortunately, geometric modeling is like shoveling sand. With every shovel you pick up a bit of dirt. The numerically imperfect nature of geometric algorithms can challenge HP PE/SolidDesigner's Boolean engine with contradictions and inconsistencies. The Boolean engine uses a perturbation method[1,2] to push the frontier of robustness. This article explains the notion of model consistency and demonstrates what can go wrong inside a Boolean operation and what can be done to come up with a correct result anyway.

Consistency of a Solid

Looking at a solid we usually believe that it is mathematically correct, that is, that the edges are exactly on their adjacent faces and the edges meet exactly at their common vertices. In reality, however, the limited floating-point accuracy of a computer introduces errors. On the microscopic level there are gaps and holes everywhere (see Fig. 1).

The tolerable amount of error is specified by the *modeling resolution*. The system will ignore gaps and holes smaller than the resolution. However, some geometric algorithms, such as the various intersection calculations, tend to magnify errors in certain geometric configurations. This means that given an input where all errors are within limits, the result can be inconsistent in the context of the solid and prohibit the successful completion of the requested Boolean operation.

Solving the Numerical Puzzle

One area in the Boolean operation that is particularly vulnerable to numerical inconsistencies is the intersection graph construction. The graph construction assumes that all intersections of curves defined on one of two intersecting surfaces are also on the intersection track (here the term *on* means closer than the resolution). This is no problem if the surfaces are reasonably orthogonal. However, for intersections between tangential or almost tangential surfaces, a small error in the orthogonal direction of a surface implies a larger error in the direction of the surface, and this assumption becomes false.

Fig. 2 shows a shallow intersection between the two surfaces sf1 and sf2 and the intersection with sf2 of a curve (cv) contained in sf1. The curve/surface intersection point (small colored triangle) has, because of the small distance (epsilon) between cv and its containing surface sf1, moved farther away from the surface/surface intersection track (colored line) than the resolution permits. The smaller the angle β the larger the distance d from the intersection track and hence the larger the inconsistency.

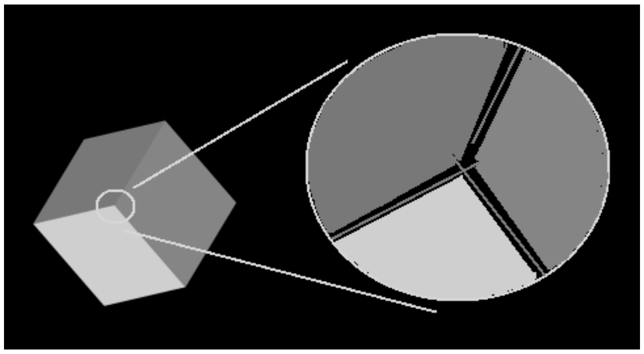

Fig. 1. In solid models edges seem to be exactly on their adjacent faces and meet exactly at their common vertices. In reality, because of the limited floating-point accuracy of a computer, on the microscopic level there are gaps and holes everywhere.

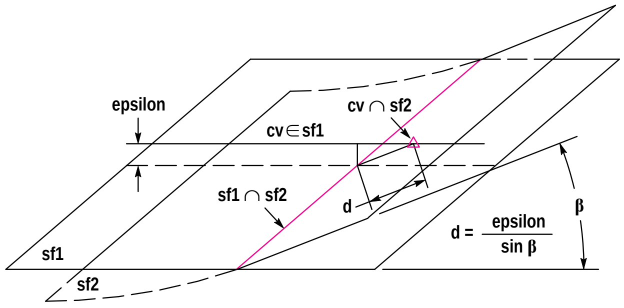

Fig. 2. A shallow intersection between the two surfaces sf1 and sf2 and the intersection with sf2 of a curve (cv) contained in sf1.

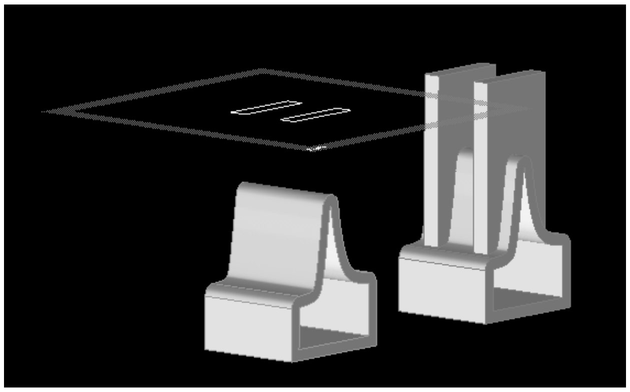

Fig. 11. A body and a profile set up for an extrude-to-part operation. To the right is the result of the operation.

extrude-to-part operation. Only the intersection graph where the extruded profile hits the body is used to build the result. To the right is the result of the operation.

Usually the extruded profile would exit the body at the bottom, producing a second intersection graph.

Reflection of Solids. Another unusual Boolean application is the reflection of solids at a plane. Fig. 12 shows a body with a green reflection plane set up. At the right is the result of the reflect operation.

This operation can be simulated with regular Boolean operations by copying, mirroring, and uniting the left body. However, this would burden the Boolean engine with difficult tangential intersections. Instead, the reflect command intersects the left body with the reflection plane to obtain an intersection graph which can be used to glue the left body

Fortunately, there is a method called perturbation than can come to the rescue in situations like this. It solves the inconsistency by moving the curve/surface intersection point along the curve until it is closer than the resolution to the surface/surface intersection track. In Fig. 2 the point will be moved to the left. When the intersection point is moved, a new error is introduced because the point is moved away from sf2. However, the overall error is reduced so that it no longer exceeds the resolution.

The perturbation method can be applied to similar situations in which even the number of intersections has to be corrected. The difference in number is a result the freedom algorithms have below the resolution. They may return *anything* in the range of the resolution.

Two Curve/Surface Intersection Points with One Surface/Surface Intersection Track. Fig. 3 shows a geometric configuration in which the intersection between sf1 and sf2 yields one intersection track (colored line) but the intersection of the curve contained in sf1 with sf2 gives two intersection points (colored triangles) which are farther than the resolution away from the track. The perturbation algorithm moves both points inwards (horizontal arrows) and contracts them into a single point (black triangle) which is closer than the resolution to the intersection track (colored line).

Two Surface/Surface Intersection Tracks with One Curve/Surface Intersection Point. Fig. 4 shows a geometric configuration in which the intersection between sf1 and sf2 yields two intersection tracks (colored lines) but the intersection of the curve contained in sf1 with sf2 gives one intersection point (colored triangle) which is farther than the resolution away from the tracks. The perturbation algorithm splits the single intersection into two and moves them outwards (horizontal arrows) until both are closer than the resolution to an intersection track (colored line).

References

1. H. Edelsbrunner and E. Mucke, "Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms," *Proceedings of the 4th ACM Symposium on Computational Geometry*, June 1988, pp. 118-133.

2. C. K. Yap, "A geometric consistency theorem for a symbolic perturbation theorem," *ibid*, pp. 134-142.

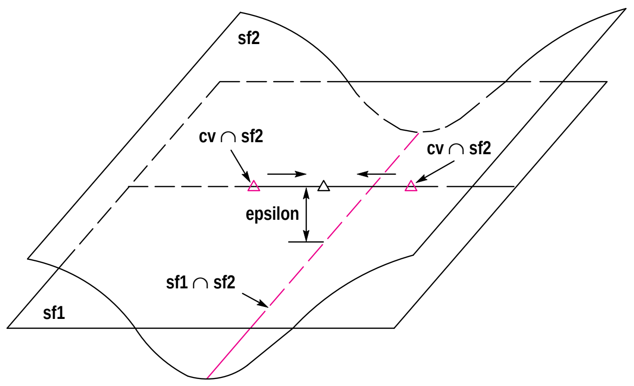

Fig. 3. A geometric configuration in which the intersection between sf1 and sf2 yields one intersection track (colored line) but the intersection of the curve contained in sf1 with sf2 gives two intersection points (colored triangles) which are farther than the resolution away from the track.

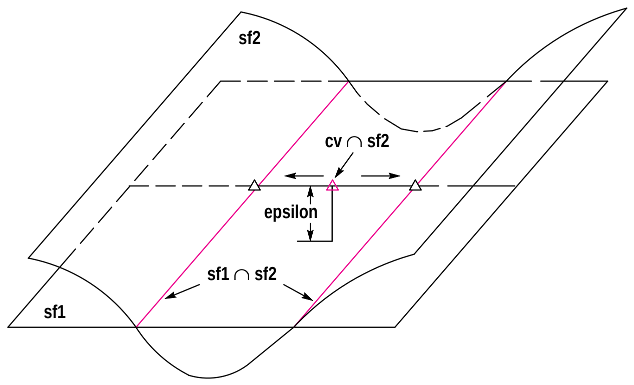

Fig. 4. A geometric configuration in which the intersection between sf1 and sf2 yields two intersection tracks (colored lines) but the intersection of the curve contained in sf1 with sf2 gives one intersection point (colored triangle) which is farther than the resolution away from the tracks.

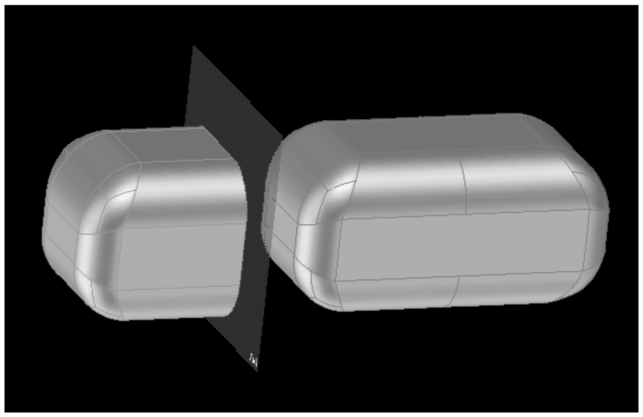

Fig. 12. A body with a green reflection plane set up and, at right, the result of the reflect operation.

and its mirrored copy together. The intersection with the mirror plane is nicely orthogonal and relatively easy to perform compared to the tangential intersections.

Acknowledgments

The development of the Boolean algorithms involved many people. Special thanks to former kernel development team members Hermann Kellerman and Steve Hull and project manager Ernst Gschwind.

Reference

1. M. Mantyla, *An Introduction to Solid Modeling*, Computer Science Press.