This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2018-7100C

Reduced Topology B-Reps

Paul R. Stallings, Ph.D.

*Sandia National Laboratories, Albuquerque, New Mexico, U.S.A
prstall@sandia.gov

Abstract. This paper presents a reduced topology structure to support boundary representation, B-Rep, modeling and explains its advantages and how it is sufficient to produce all the information in existing B-Rep structures. Specifically, the reduced structure removes lumps, shells, loops and coedges in favor of automatically deriving their information from the remaining data. The current state of B-Rep modeling is explained along with a new reduced topology structure that accommodates mixed dimensional, non-manifold, seamless B-Reps. In addition, a section has been added that explains how to handle the difficulties that seamless modelers and closed surfaces bring to B-Rep modeling.

Keywords: B-Reps, CAD formats, Topology Data Structures, Faceting, Seamless modelers.

*Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

1 A Brief History of B-Reps

Computer aided design, CAD, started with wire-frame data, then surface modelers were created followed by solid modeling. Solid modeling started with making primitive shapes such as blocks, cylinders and spheres and then combining them in a constructive solid geometry, CSG, tree. CSG trees are easy to work with but have serious limitations that B-Reps solve. However, B-Reps are relatively difficult to work with, hence the need to simplify the traditional B-Rep structure.

Currently B-Reps are the standard for geometric modeling. However, they have historically had several limitations. When first described, B-Reps were mostly manifold solids, and the first B-Rep versions did not work well with mixed dimensions such as wire frame data and surfaces. Moreover, only recently have commercial B-Reps started to incorporate zero-dimensional data such as clouds of points.

The primary limitations of B-Reps are in their boundary only view of a model and in their relatively heavy finite piecewise nature. Hence, medical data from MRI's and CAT scans are not well represented with a B-Rep and complicated structures such as weaved cloth or the many voids and channels in the fine grain structure of a human bone are very memory intensive if represented with a B-Rep. This paper does not claim to address these limitations of B-Reps other than to claim that by reducing the structure memory limits are extended making objects such as a golf ball with its many dimples or a speaker grid of holes use less memory. A good reference for a more complete history of B-Reps can be found in [1].

2 Current B-Rep Data Structures

Different geometric modelers on the market today use different data structures and standards like STEP attempt to select a compromise between them for translating data between different B-Rep modelers. Hence, let us start by describing the main parts that are common to most modelers and then pointing out the differences that exist between them. Most modelers have as their top-level entity a Body, which can be a three-dimensional solid body, a two-dimensional sheet body, a one-dimensional wire body, a zero-dimensional cloud of points, or any combination of these four dimensions. For the sake of mathematical completeness let us define the subsets of three-dimensional Euclidean space, $\mathbb{R}^3$ that a B-Rep may define.

3 Definition of a B-Rep definable subset of $\mathbb{R}^3$

Any closed and bounded subset of $\mathbb{R}^3$, whose boundary consists of a finite number of differentiable faces, edges and points may be defined by a B-Rep. However, for the definition to have meaning we need to define a face and edge along with a surface and curve. We will assume that a point in $\mathbb{R}^3$ is well understood.

A surface is a function from a coordinate aligned rectangle in $\mathbb{R}^2$ to $\mathbb{R}^3$. The function must be differentiable and one-to-one on the interior of its domain. If the function is not one-to-one on the boundary of its domain, then only three cases are

admissible. One, a whole side of the domain's rectangle can map to a single point, in which case the surface is said to be singular at that point. Two the left side of the surface's domain may map horizontally to the right side, in which case the surface is said to be closed in U, which is traditionally used to denote the X coordinate of the domain of a surface. Three, the top of the surface's domain may map vertically to the same point as the bottom of the domain, in which case the surface is said to be closed in V, which is traditionally used to denote the Y coordinate of the domain of a surface. These limitations on how a surface's boundary may map to itself restricts all admissible surfaces to have genus zero as in the case of a plane, sphere or cylinder, or genus two as in the case of a torus.

A curve is a function from a closed line segment in $\mathbb{R}^1$ to $\mathbb{R}^3$. Like a surface a curve's defining function must be differentiable and one-to-one on the interior of its domain. If the two end points of its domain map to the same point the surface is called closed.

Surfaces, curves and points define the geometry of a B-Rep and are used by faces, edges, and vertices to define the shape and / or location of their part of the boundary. However, a face and an edge do not have to use their whole surface or curve. In surface modeling, subsetting by giving a set of edges to define its boundary to be something other than a rectangle in its domain is called a trimmed surface. However, in B-Reps surfaces and curves may not only be trimmed but they may also be shared with more than one face or edge.

4 The Topology Data Structures

Most modelers use something close to the following data structure.

Body → Lump → Shell → Face → Loop → Coedge → Edge → Vertex

Figure 1

The purpose of this paper is to explain how to eliminate Lumps, Shells, Loops, and Coedges. However, first let us define each of the entity types.

A vertex's geometry is a single point. Vertices make up the boundary of edges, or stand alone to define a point in space or zero-dimensional part of the B-Rep.

An edge's geometry is a subset of a curve. Edges may lie on boundary of one or more faces, or stand alone to define a wire edge or one-dimensional part of the B-Rep. Moreover, some edges may lie in the interior of a face.

A coedge's geometry is a parameter curve, which is like a curve but is a function from a closed interval in $\mathbb{R}^1$ to $\mathbb{R}^2$, where $\mathbb{R}^2$ is the parameter space, or range, of the surface of the face to which the coedge belongs. Coedges are the glue of most modelers in that they point to the previous and next coedge in their loop, and they point to their partner coedges on the same edge. Moreover, coedges also contain a flag that tells how they are oriented with respect to their edge.

Loops do not have geometry. Instead a loop consists of a cyclically ordered set of coedge(s) that make up a connected component of the boundary of a face. Loops may either be an outside or inside loop, and in some modelers all faces have only one outside loop and any finite number of inside loops. In modelers that support closed surfaces, a face may have more than one outside loop.

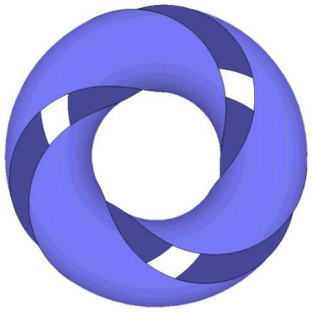

Figure 2
An Example of a Torus Face with two Outer Loops Formed By two (2,3)-Torus Knots

A face's geometry is a subset of a surface. Faces make up part of the piecewise differentiable boundary of a lump. In addition, faces may have a flag that indicates if the surface's normal, or cross product of the U and V derivatives, points into or out of the body. Faces may also have a flag that tells how many sides of the face have material on them. A face of a solid body has material on only one side. However, a face of a sheet body has material on zero sides. It is possible to also have what is called a membrane face that has material on both sides.

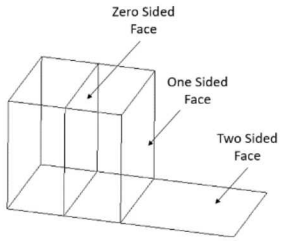

Figure 3
An Example of Zero, One and Two-Sided Faces

Shells, like loops, do not have geometry. Instead a shell consists of an unordered set of face(s) that make up a connected component of the boundary of a lump. Also like loops, a shell may either be an outside or inside shell. However, unlike loops, all lumps only have one outer shell and any finite number of inside shell(s). Inside shells are sometimes called voids. In addition, there are two types of shells, open shells and closed shells. A closed shell separates $\mathbb{R}^3$ into an unbounded component and one or more bounded components.

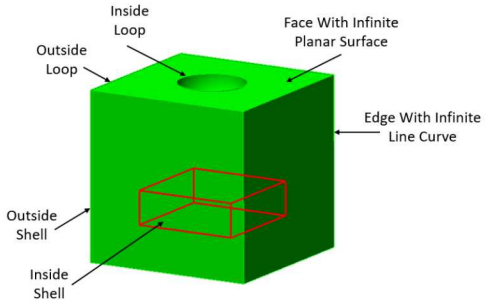

Figure 4
An Example of Different Topology Entities

Lumps, like shells and loops, do not have geometry. A lump is intended to represent a connected region of space in the B-Rep. If the shells are closed shells, then the connected region of space is a three-dimensional part of $\mathbb{R}^3$. If the lump has just one open shell, then the region of space is a two-dimensional part of $\mathbb{R}^3$. To accommodate one-dimensional regions of space a Lump also may contain an unordered set of edges. The location of where a set of wire edges is stored in a geometric modeler is somewhat ambiguous in that they could be at the body, lump, or shell level.

Bodies also do not have geometry. Instead they consist of an unordered set of lumps. Since a single, isolated point is in and of itself a connected component of a B-Rep, it can be a lump on its own. However, to avoid creating a heavy data structure one can add an unordered set of points to the definition of a body along with its set of lumps.

5 A Few Notes on Modeler Differences

The above data structure is what one will find in the ACIS geometric modeler owned by Spatial, with the exceptions of how clouds of points and wire bodies are handled. Another commercial modeler, called Parasolid, has a similar data structure, that differs in one important way in that instead of lumps it has regions.

Regions are connected parts of $\mathbb{R}^3$ after the boundary of a B-Rep has been subtracted from the space. Hence, a block with a void spherical shell inside it has one lump and two shells. However, in Parasolid a block with a void spherical shell inside it has three regions, the unbounded region, the inside void region and the solid region between the shells.

Another major difference between the modelers is how they handle close surfaces and curves. Parasolid, and SGM are both seamless modelers, in that they let both surfaces and curves be closed and they do not require edges at the seams of closed surfaces and vertices at the ends of closed edges. ACIS adds seams on non-simple surfaces and vertices at the ends of closed edges. However, it also permits closed surfaces and curves. However, the geometric modelers in Catia and Pro-E, now called Creo, do not permit closed surfaces or curves and hence split them up so that their surfaces and curves are one-to-one on their whole domain. Moreover, Catia requires that its surfaces and curves be not just differentiable but G2 also.

Because there is disagreement on the existence of seams and vertices on closed edges most modelers default to adding them when outputting STEP files so that modelers that require them can read their files in. Since seams add to the number of faces and sometimes edges, and since requiring them can be a burden on reading a file from a system that does not require them, SGM has opted to leave seams out but like most systems adds them if requested.

One other major difference between modelers is which part of the B-Rep structure is used to define the location of an edge, either the three-dimensional curve as in ACIS, Parasolid, and SGM or the two-dimensional coedge's curve as in Catia and Creo. In the IGES standard both may be given and a flag is also given to tell which one to trust. In the STEP standard the three-dimensional curve is the primary definition.

6 Reducing the Topology Data Structure

In theory all that is required to define the subset of space that a B-Rep occupies is a set of Faces, Edges, and Vertices, along with their surfaces, curves and points. The set of faces and a set of isolated points can be contained in a body. Hence, we can remove lumps, shells, loops, and coedges. In most CAD programs lumps are not very important. However, in meshing programs like Cubit from Sandia National Labs, lumps, called volumes, are the primary data structure, since they represent a connected subset of $\mathbb{R}^3$ that needs to be meshed. Hence, we are going to explain how one may derive the defining data of the lumps, shells, loops and coedges but in the new reduced topology modeler SGM, that is being developed at Sandia National labs, we left volumes in the data structure.

First let's explain how one can derive the defining data of Lumps, Regions, and Shells from the minimal data of a body's faces, edges, and vertices, and then we will explain how one can derive the defining data of loops and coedges from the minimal data.

7 How to Find Lumps, Regions and Shells

Assuming we have a data structure where faces know their edges, edges know their vertices, and the other way around, vertices know their edges, and edges know their faces, we can start by creating a graph of the faces where the vertices of the graph are the faces and two faces are adjacent in if they share an edge or vertex. Once such a graph is formed it is an easy matter to find the connected components of the graph. Each component is a shell.

Once the shells have been found, each shell can be tested to see if it is open or closed. A shell is open if all its edges are either double sided or belong to more than one face. The next step is to place a partial order on the shells based on containment, which is to say that shell B is less than shell A if A is a closed shell and B is disjoint from the unbounded region of space that is left in $\mathbb{R}^3$ after removing A from $\mathbb{R}^3$. To test to see if B is contained by A take any point P on B and intersect a ray starting at P with A. If all the intersections are not tangent intersections and the number of intersections is an odd number, then B is contained in A. If tangent intersections are found then either one may pick another point or direction for the ray or one can use more advanced methods to tell if the ray passes from one side to the other of the shell at the intersection or if it just tangentially intersects the shell without changing sides. In SGM such intersections were resolved by selecting the first ray direction to be the vector (0,0,1) and following ray directions to be the unitized vectors of the form (cos(n),sin(n),cos(13n)) as the integer n grows without bounds cos(n) and sin(n) take on a dense set of disjoint values in the interval [-1,1]. Hence, given the finite nature of a B-Rep eventually, most often in the first or second case, one ray will intersect the shell non-tangentially at all its intersections.

Once the partial order has been found the next step is to find all the maximal elements. We will call the shells that are maximal elements in the partial order first generation shells. Then remove the maximal elements from the partial order and find maximal elements in what remains, these shells are called second generation elements. Second generation elements that are contained in first generation element are called the children on the first-generation elements. Since there are a finite number of shells one can find the generation of each shell and its children.

Given these results the first-generation shells are outside shells, and for each first-generation shell their children go with them to form a lump. The third-generation shells are also all outside shells and along with their children they form lumps that lie inside the voids of other lumps. Continuing in this way each odd generation along with their children define the inside and outside shells of the lumps of the B-Rep. Given the shells and lumps of a B-Rep one can find the regions by creating one unbounded region with all the first-generation shells, a region for each lump and a region for each inside shell minus the lumps that are contained inside it. If a shell has membrane faces, then more care must be taken to find the cells of each lump, which is a topic that will be covered after we explain how to order the coedges of a loop.

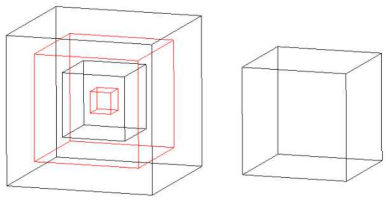

Figure 5
Black Shells are Outside Shells. Red Shells are Inside
All Together Three Lumps and Five Shells

8 How to Find Loops and Coedges

Much like finding shells, the first step in finding loops is to create a graph of the edges of a face and find the connected components, where two edges are adjacent if they share a vertex. Once the loops are found the next step is to order the edges so that they go around their face in counter clockwise order. If the flipping flag on the face is set, then the edges need to be ordered in clockwise order. Here is where one needs a flag for each edge and face pair to tell which side(s) the face is on. Without this flag the subset of a surface that a face uses is ambiguous. Hence, in SGM each face maintains a map of is edges to a flag that tells if the face is on the left, right or both sides of the edge. An example of a case where this flag is needed is a circular edge on a sphere. Without the flag one cannot tell which side of the edge the face is on. In modelers with coedges the coedges are assumed to have the face on their left and a flag is maintained to tell if the coedge's direction matches its edge's direction. By moving this information from the coedge to the face, coedges can be eliminated from the modeler.

Nevertheless, there is still a need to be able to return a cyclically ordered list of edges of a loop along with a flag for each edge as to if the face is on the left or right. If the edge is a seam or interior to the face, and hence the face is on both sides of the edge, then the list of edges returned for the loop should contain the edge twice, once for each side. In modelers with coedges, there would be two coedges on the edge in the loop for a double-sided edge. In general ordering the edges is a simple matter of considering the side flags and walking around the loop using the vertices to go from one edge to the other. The problem of ordering the edges become more difficult in the case that one encounters a vertex with more than two edges from the face, or in the case of closed edges more than two end points of edges from the face.

In the case that more than two end points of edges from the same face share the same vertex, then the ends of the edges coming into and out of the vertex need to be ordered cyclically around the face's surface normal, and as one marches around the loop one needs to take the next edge in this local cyclical ordering in the clockwise direction as the next edge in the loop in the counter clockwise direction. Once done the ability to return a list of lists of edges along with the side flags replaces the need for loops and coedges. One detail is that double sided edges need to count as two edges coming into a vertex and each side needs to be treated as a different edge when ordering the edges of a loop.

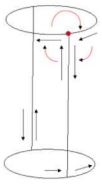

Figure 6
Red Arrows Show How to Order the Coedges About the Red Vertex

The partner pointers of a coedge to the other coedges of their edge are replaced by making sure that each edge knows the faces to which it belongs. As for the parameter curves on a coedge, in modelers that use the three-dimensional curve as their primary definition all parameter curves can be generated from the three-dimensional curves as needed. However, if a modeler supports seams, in other words closed surfaces and curves, finding surface parameters is a not trivial problem. Hence section 10 has been added to tell how to find facets in seamless modelers.

10 Finding Cells

Cells are a little used topology that is most often derived from other things as needed. Three-dimensional cells are best described as air tight compartments inside the B-Rep. Finding them is analogous to ordering the edges in a loop in that one marches from face to face across edges with the difficult case coming when an edge is found that belongs to more than two faces. In the case that an edge belongs to more than two faces, then the faces need to be cyclically ordered around the edge with respect to the right-hand rule with respect to the direction of the edge. Given this ordering and the side flags of the edges on the faces one can determine which face is the adjacent face to a given face in the same cell. It should be noted that when ordering the faces around an edge one should use the facets of each face instead of the tangent outward directions on the surfaces at each edge because the facets will resolve ambiguities in the case of tangent intersections of the surfaces at the edge in question.

10 Finding Facets in a Seamless Modeler

In general modelers like SGM, Parasolid, and ACIS all use the three-dimensional curves of an edge as the primary geometry definition of the edge. However, there exists some algorithms that are best solved in the two-dimensional parameter space of a surface, one such algorithm is the creation of facets for a face. Hence, this section on how to work in parameter space without coedges has been added by providing an outline of an algorithm that creates facets for a face, without coedges, or parameter curves in a seamless modeler. The following are the steps used in SGM to create facets to display faces and edges.

1. Facet all the edges of the part.
    a. Facets of edges should be faceted to tighter, or at least as tight, a standard as the triangles of a face are faceted. Facet tolerances may include angle tolerances between edge facets, or between surface normals, and / or maximum edge lengths, and / or cord heights.
    b. Each of the surfaces for each of the edges should be checked to make sure that the surfaces normal differences are as tight or tighter than the surface normal tolerance used on the face triangles.
    c. The parameter values of each facet point on each edge to should be checked to see if the facets cross a seam of surface. If it does, then all such crossing should be found and the edge should be faceted independently for each seam crossing free interval. By refaceting the edge, instead of just inserting a point, one lowers the chance of creating a very small facet edge.
    d. When refining the edge facets, to meet the surface normal tolerance, care should be taken to not infinitely refine the edges at their end points that may lie at a singularity of a surface.

2. Using the method described in section 8, find the list of lists of cyclically ordered edges, along with their left or right face side flags.

3. For each loop take the faces of each edge, reversing them if the edge has a right face side flag, to create one cyclically ordered set of facet points for each loop of the face.

4. Once the facet points associated with each loop has been found, find the corresponding UV parameter space values on the face's surface for each point. If the surface is closed in U or V, or has singularities on its boundary, then the inverse problem of evaluating the surface's defining function is ambiguous. To aid in resolving this ambiguity one should create a special inverse function that takes in the XYZ point in $\mathbb{R}^3$, along with the edge that it came from, and the face side flag returned for it in step 2. Given all this information it is possible to resolve all the ambiguates, and like the topology reconstruction functions, this code should be written one time and used by the rest of the modeler when finding inverses of points on the boundary of a face.

5 At this point the boundary facets may now be considered to lie in the two-dimensional parameter space of the face's surface. If a surface normal tolerance of less than π radiant is used, then it may be assumed that any boundary facet that jumps more than half way across the parameter space of a closed surface, in the direction that it is closed, must cross the seam in that direction. Moreover, since care was take in step 1c to make sure that the facets of an edge always have a facet point on the surface seams no point must be added to the facets for the seam crossings. However, all such crossings should be recorded and then linked together in step 6. In addition, any singularities should also be noted.

6 At this point the boundary facets of a face have been laid out in parameter space and all the crossings of seams and singularities have been found. Hence, to form a set of closed polygons for the facets of each loop in parameter space, the linked list of facet points needs to be severed at each crossing and singularity and extra facets, meeting the facet tolerances, need to be added along each seam and along the parameter space that is spanned by a singularity. Figure 7 shows the faces of the toroidal face from Figure 2 that was bounded by two torus curves that each wrapped around the major radius of the torus two times and the minor radius of the torus three times. The red arrows are from one of the two loops, and the blue arrows are from the other, indication the direction of the boundary facets. Step 6 is to use these extra seam facets to link up the ends of the arrows, to the next start of an arrow found by tracing around the boundary of parameter space in a counter clockwise direction. If the flipping flag of the face is set then everything is in done in the clockwise direction.

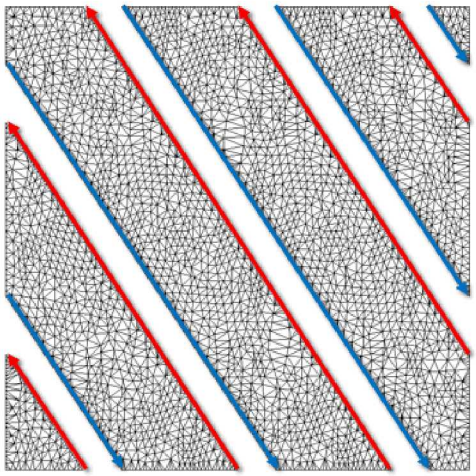

Figure 7
The Faces of the Figure 2 Toroidal Face

7 Once the seam and singularity crossings have been taken care of what is left is a finite set of polygons in parameter space. In the case in figure 7 there are six polygons. It should be noted that the polygons might not be simple polygons in that they might form branched graphs if a face has a loop that contains a part that tangentially touches it as shown in figure 8. Nevertheless, non-simple polygons will be sufficient for the next step and they may be found by starting at any node in the linked list of facet points and tracing around until one returns to where they started, keeping doing this with nodes that have not been used, until all the polygons have been found.

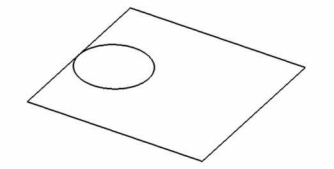

Figure 8
An Example of a Branched Loop

8 The next step is to find the outer polygons by forming a partial order of containment of one polygon inside another like the shells were found in section 7. Once found, then the polygons should be broken up into groups, each with one outer polygon and its children.

9 Given a set of polygons with holes the next step is to triangulate them, which can be done in n*log(n) time. However, SGM used an algorithm found in [2] that might be less than optimal order but make better triangles.

10 Once a base triangulation has been found, the next step is to refine the triangles to meet the desired tolerance. This can be done by adding a mid-point to each interior edge, noting that the boundary facet edges already meet or exceed tolerance, and splitting the two triangles of the interior edge in half at the edge's mid-point. The edges should be ordered and taken from worst to best, doing incremental Delaunay flipping as each edge is split.

11 The only other consideration is that one might want to scale the parameter space to bring the direction of least curvature closer together so that the facets of a cylinder or cone will favor long thin facets that run in the direction of zero curvature.

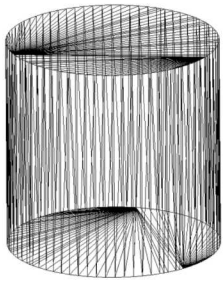

Figure 9
The Display Facets of a Cylinder

11 Conclusions

Hence, if one moves the edge side flags from the coedges to the face, and sets up a data structure where faces know their edges, edges know their faces and vertices, and vertices know their edges, then one can derive all the defining information of the regions, lumps, cells, shells, loops and coedges. However, one may still ask why would one want to do this? There are several reasons why.

The first reason, and the inspiring reason, to reduce the topology in making a new modeler comes from solving the inverse problem of converting a B-Rep to a set of facets to display. Given a set of facets, how does one find a light weight B-Rep for them? The complete algorithm of how one converts facets to B-Reps is a subject for another paper. However, roughly one first splits the facets up into the faces by finding co-planar, co-spherical, co-cylindrical, … facet points on a set of adjacent triangles and then finds the boundaries of these subsets of triangles and similarly finds co-linear, co-circular, … facet points to fit curves to the boundary. Doing this results in a set of faces, edges and vertices for which one needs to construct a B-Rep. By having a reduced topology modeler, this otherwise difficult job is done after finding the faces, edges and vertices.

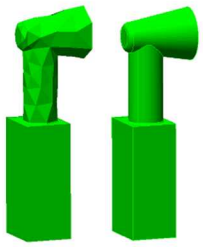

Figure 9
An Example of Converting Facets to a B-Rep

The second and perhaps more important reason is that by building the automatic finding of lumps, shells, loops, and coedges into the topology of a modeler one eliminates the need to maintain these data structures all throughout the code in other algorithms such as blending, Booleans, shelling, local operations, etc. A simple example to illustrate this point is to consider what it takes to merge out two seams added to a cylinder. In this case the linear edges each belong to two loops and four coedges. In this case, in a modeler with loops and coedges, relinking up and moving everything is relatively difficult compared to the ease of just removing the two edges from their faces and vertices, and then moving the edges from one face to the other, at which time everything else is automatically taken care of for you. By just having to add or remove faces or edges from their volumes or faces all the other topology manipulations algorithms also become easier to write, cutting the time and cost of developing a modeler significantly.

A third reason is that by consolidating the topology manipulation of lumps, shells, loops and coedges one eliminates the almost inevitable code reuse in existing modelers. Moreover, by reusing the same topology manipulation code in one place bugs are found faster and fixed one time instead of many times. In addition, it should be noted that the automatic finding of lumps, shells, loops and coedges needs to be written anyways in most modelers that support the ability to stitch faces together, and the ability to cover edges with a given surface.

A fourth reason is that by reducing the topology some of the ambiguity of where to put wire edges and clouds of points resolves itself. In SGM wire edges are added to the volume or lump, and the cloud of points is added to the body along with its set of volumes.

Hence, what has been shown is how to define a reduced B-Rep data structure that supports non-manifold bodies of mixed dimensions from zero to three. In addition, one can add interior faces, edges or vertices to define parts of a body that do not change the subset that the B-Rep defines.

References

1. Kevin J. Weiler, “Topological Structures for Geometric Modeling” Doctor of Philosophy dissertation Rensselaer Polytechnic Institute.

2. Gang Mei, John C. Tipper and Nengxiong Xu, “Ear-clipping Based Algorithms of Generating High-quality Polygon Triangulation”