Extraction of Boundary Representation from Surface Triangulations*

Petr Krysl[†] Michael Ortiz[‡]

Abstract. Many computational science tools employ finite element meshes as discretizations of the geometrical domains, and automatic mesh generation has become an indispensable part of the discretization process. Boundary representations (BRep) of solids are the means of describing the geometrical model to the mesher, thus enabling the generator to proceed without user intervention. Significant effort has been devoted in the past to BRep construction in the framework of solid modeling systems. In this paper we consider the task of converting a tesselation (triangulation) of the surface of a solid into a BRep, and we propose a robust and efficient set of algorithms for this purpose. Applications include, among others, remeshing of finite element discretizations during simulations involving not only geometric distortion but also changes in topology (coalescence and fragmentation of solids, flow, and so on).

Keywords: Surface triangulation, boundary representation, non-manifold topology, automatic finite element mesh generation

Introduction

Automatic mesh generation, both on three-dimensional surfaces and in solid volumes, relies on an unambiguous geometrical description such that questions of the type "Which faces bound a given three-dimensional region?" and

*Submitted to the International Journal for Numerical Methods in Engineering

[†]Staff scientist, California Institute of Technology, `pkrysl@atlantis.caltech.edu`

[‡]Professor of Aeronautics and Applied Mechanics, California Institute of Technology, `ortiz@atlantis.caltech.edu`

"On which side of this face is the solid?" posed by the meshing module to the geometry (CAD) module can be answered without human intervention [1]. The established solution is to make the geometry an attribute of a topological model [2, 3].

Boundary representation (BRep) is one of the topological description schemes in common use. The BRep should be sufficiently general to cover so-called non-manifold (meaning non-2-manifold) topologies, which include such important and common cases as layered composites, welded or glued objects, cracked geometries, and representation of computational models with non-homogeneous manifold dimensions (for example, textile re-inforced casts or soil structures with impermeable sheet screens). An overview of representation schemes for non-manifold objects has been given by Requicha and Rossignac [4].

Solids are in many important cases defined by a tesselation of their boundaries and perhaps also of their interiors. Consider finite element simulations of large deformation of solids, for instance during extrusion, casting or high-velocity impact. The initial finite element mesh was probably generated off the CAD model of the solid. The geometry might have been defined in the form of quadric or spline surfaces. Once substantial deformation develops during the simulation, the finite element mesh becomes too distorted, and in order to preserve accuracy, remeshing is needed. Remeshing the original geometry is rarely an option in large deformation analysis, and the geometry of the solid is in the deformed state defined by the finite element mesh itself. But not only that, it is possible that the *topology* of the solid changes due to cracking or fragmentation, wetting (e.g., two grains of melted plastic become one), or simply because creases are removed from or introduced to the external boundary of the solid by the deformation process itself (e.g. through a contact with the die or rolling wheel). One way of dealing with this kind of change is to modify the original BRep, and a successful implementation of such a procedure has been reported by Pandolfi and Ortiz [5]. Another option is to build the BRep from scratch, and it is our main goal in this paper to show how it can be done in an efficient and robust way.

Most papers dealing with boundary representation in CAD environments proceed from the assumption that the geometry is defined either through Constructive Solid Geometry or other operations (sweeps, skinning, ...), and delve into issues such as classification of surface and curve intersections; see, for instance, References 6–9. The PhD thesis of Weiler [10] is especially notable in that both a very general topology representation of non-manifold geometries (radial-edge datastructure), and a set of operators (Euler operators) have been proposed.

Several papers dealing with issues of boundary tesselation analysis and verification for meshing of three-dimensional volumes have appeared in the past few years. Lo proposed a set of heuristics for defining the initial front for advancing front meshing of solids with tetrahedral elements [11]. Some of the algorithms were designed for manifold surfaces only (check for closedness, orientability, containment), and no true boundary representation which could be used in remeshing is built. In a later work, Lo presented a collection of techniques for analysis and verification of triangulated boundary surfaces of solid objects [12]. The main limitations of Reference 12 are (i) non-manifold surface configurations (and in particular sheet faces) are not handled robustly, and (ii) BRep topology is not built in the completeness required for automatic surface and volume remeshing.

Löhner describes an advancing front re-gridding technique which works directly on surface triangulations [13]. The topological feature recognition is limited, in fact edges are recognized, but not linked into bounding loops, and faces are not organized in any way to bound three-dimensional volumes. Geometrical criteria for edge and vertex detection are essentially those adopted in this paper.

There are some parallels between the present approach and the work of Chivate and Jablokow, who proposed a technique for construction of BReps of solids bounded by piecewise quadrics [14]. However, the BRep is not strictly built off a surface tesselation, and the solid models are limited to manifold geometries (the modified winged-edge data structure is used). Also, since the input data is gathered from coordinate-measuring machines, neither voids, nor internal interfaces are considered.

Given a BRep topological model and a set of primitive operators for manipulation of this model (Weiler's radial edge and the associated Euler operators being a particularly prominent example), any collection of simplices can be transformed into a valid BRep [10], because simplices of dimension 0, 1, 2, and 3 have direct counterparts in the topological model (vertices, edges, faces and solids). However, using primitive operators directly on the simplicial mesh is not likely to be efficient, and it certainly is not the only approach. Therefore, our goal in this paper is to find a robust and efficient algorithm for constructing a boundary representation of solids given through their tesselated boundaries, which is suitable for subsequent processing in computational science applications.

The paper is organized as follows. We start off with some definitions and notations in Section 1, and in particular we introduce our topological model and compare it to other established BRep models. Section 2 describes the overall flow of data in algorithm, and Section 3 describes the pre-processing step for orientation of triangulations. Sections 4 to 7 deal with the extraction of topological faces, edges, loops. The detection of sheet, closed manifold, and non-manifold shells is described in Section 7, as is the creation of solid regions. Pseudo-code is given for all the major algorithms. Ways of re-defining the tesselated geometry by subdivision surfaces are briefly discussed in Section 8. Model verification through topological invariants and geometrical checks is discussed in Section 9. The paper concludes with a number of examples.

1 Definitions and notation

Surface triangulation consists of triangles (2-simplices), which are either disjoint, or share a vertex (0-simplex) or an edge (1-simplex). Therefore, the surface triangulation is a simplicial complex [15]. We assume there are no dangling 0-simplices (vertices) or 1-simplices (edges) in the input, but to accomodate them in the data structures is straightforward.

We define the **oriented surface triangulation** as a triangulation in which the number of triangles joined along their edges in a 2-manifold arrangement *compatibly* is maximized. (Triangles $\triangle ABC$ and $\triangle DCB$ are oriented compatibly; see Figure 1. If the triangles were embedded in the plane of the paper, their normal would stick out.) The algorithm of Section 3 yields a (trivial) solution of this optimization problem for triangulations of orientable surfaces. If given a triangulation of a non-orientable surface, this algorithm finds just one of the possible trial states, which is not the solution of our maximization problem, in general. So far our interest in triangulations of non-orientable surfaces was rather marginal, and we have not implemented a true optimization procedure for this case.

The orientation of the triangulation induces orientation in the topological faces which refer to the triangulation as to its geometry definition.

The oriented surface triangulation is to be contrasted with a **raw surface triangulation** which consists of triangles oriented arbitrarily. Raw triangulations are often obtained from geometric (polygonal and other) models by triangulating the individual geometric entities (polygons, Bezier or spline surfaces, and others) and merging the resulting simplices using tolerances.

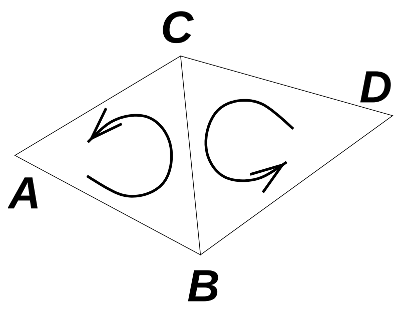


Figure 1: Orientation of triangles.

Topological vertex (TV) represents a topologically distinct point. Topologically distinct points have not necessarily distinct locations in the three-dimensional space.

Topological edge (TE) represents a curve, is non-self-intersecting, and is bounded by two vertices at each end (open edge), or its two ends coincide (closed edge). A TE is an oriented entity; the orientation is induced by the order of the bounding vertices, or, if the edge is closed, the orientation is induced by the order of the mesh edges classified on the TE. Each TE may bound one or more surfaces. Each of these occurrences is recorded as a *topological edge use* (TEU). An edge use is oriented with respect to the underlying edge, i.e. its orientation is either *same* (meaning the same as its owning edge) or *reversed.* A given TE is either a *sheet* edge, if it is used just once to bound a surface, *manifold* (read "2-manifold"), if it is used twice to bound a surface, or *non-manifold* if it is used more than twice to bound a surface.

Topological loop (TL) is a collection of TEU's which bound a given TF, and such it is an oriented entity, because each of its constituents is oriented. (More precisely, a TL is a circular list of TEU's. However, none of the algorithms in this paper require this ordering.) Any TE may be listed in a given TL through its use many times, i.e. it may be used just once in the same or in the reversed orientation, or it may be used any number of times with any orientation. One example is given in Figure 2, which shows a planar diagram of a cylinder and a Möbius strip (non-orientable surface). The surfaces are obtained by gluing together the edge **a** according to its orientation. Note that the loop in face **M** of the Möbius strip uses TE **a** twice with the same orientation. Another example is given in Figure 3. It shows a spiral surface which is strung on the outer spiral curve and the central thick-line edge. All the TE's are collected in a single loop, and the thick-line edge is used in this loop three times. Evidently, one could continue wrapping the spiral ad infinitum.

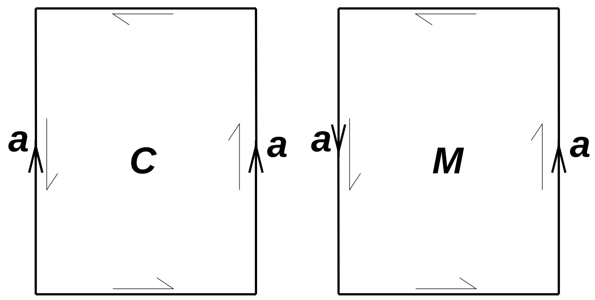


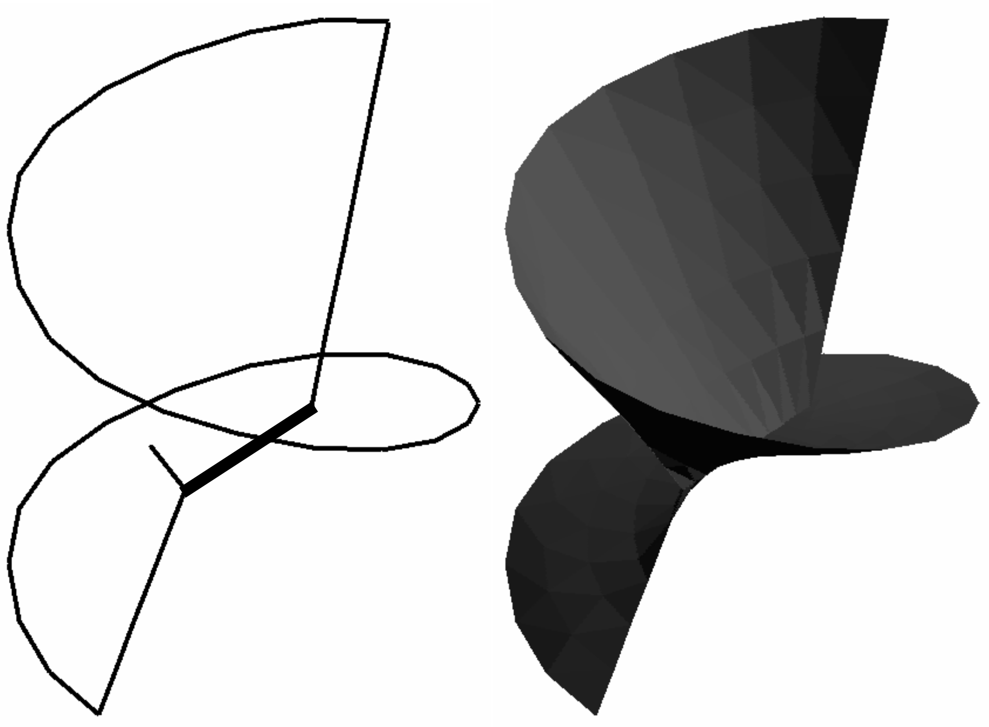
Figure 2: Use of the glued edge in a cylinder and a Möbius strip.

Topological face (TF) is a compact, connected, and oriented piece of surface. A given TF is bounded by zero, one or more topological loops. (If a TF is not bounded by any loop, we can arbitrarily place a TV on the surface, and declare the TV to represent a TL with no edges.) TF may bound up to two solid regions, which is recorded by a *topological face use* (TFU). The orientation of a TFU is defined with respect to the orientation of the owning TF, which in turn is inherited from the underlying surface triangulation (hence the use of *oriented* triangulations as input to the topology extraction algorithm); compare with Figure 4, where the TFU with orientation *SAME* bounds TR1, and the TFU with orientation *REVERSED* bounds TR2 (the orientation is shown as normals to the surface).

Topological shell Is a list of TFU's which bound a given TR. TS may be orientable and closed, in which case it may enclose some volume (i.e. a topological region). If a TS is not orientable, or is not closed it still bounds a single TR, but represents a sheet embedded in the TR. Figure 5 explains the concept of a topological shell. Faces TF1 and TF7 do not belong to the shell of region TR1, because they do not bound TR1.

Topological region (TR) is a connected volume. It is bounded by zero or more topological shells.

Figure 6 summarizes the analogy between the bounded (bounding) topological entities of the two- and three-dimensional subspaces (surfaces and solids) in our topology model. The labelled arrows indicate a functional relationship, and the numbers "1" and "N" encode symbolically the number of



(a) Topological edges. (b) Topological face.

Figure 3: Example of topology involving multiple use of a TE in a single loop.

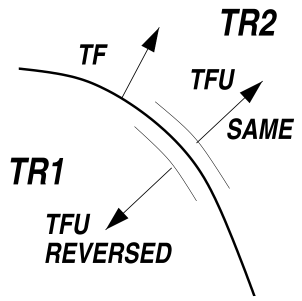


Figure 4: Orientation of a TF and its uses.

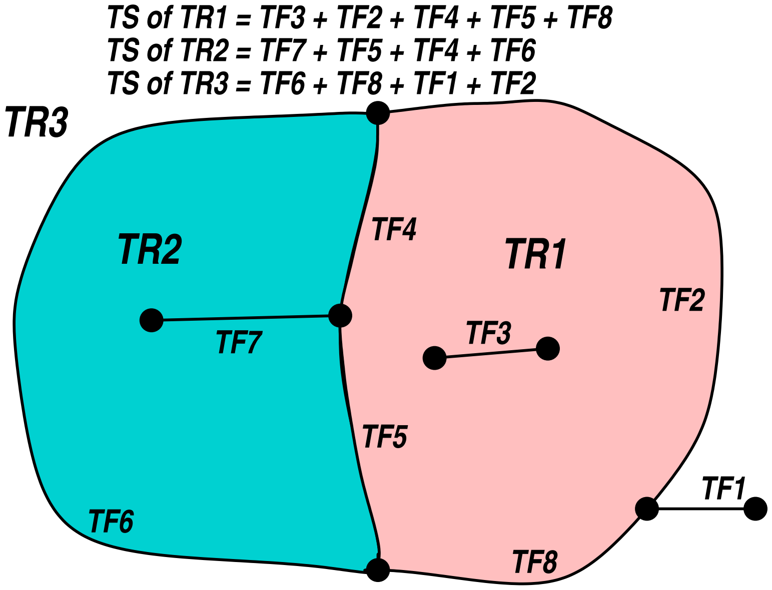


Figure 5: Explanation of the concept of topological shells. TS consists of only those TF's which bound given TR.

entities involved in a given relationship. Note the distinction between TFU's and TEU's: Any given TF can be used at most twice, whereas a TE may have any number of uses attached. This is due to the fact that a TF is a *two-dimensional* manifold embedded in R^3, so it can be folded and joined along an edge as its bounding entity an arbitrary number of times.

The acronyms used in this paper are for the reader's convenience summarized in Table 1.

1.1 Comparison with other topological descriptions

Our definition of the topological entities differs in some respects significantly from other established topology models. In particular, Weiler [10] does not consider TF's oriented entities (only *orientable*), and assigns orientation to the uses of the face. In our model, TF is an oriented entity, and orientation of its uses is defined relative to the owning TF. An analogous comparison can be made for the edges. Since the loops in our model correspond to *loop uses* in Weiler's model, there is a difference in the number of edge uses per edge: one per face in our model vs. two per face in Weiler's model. Also, we

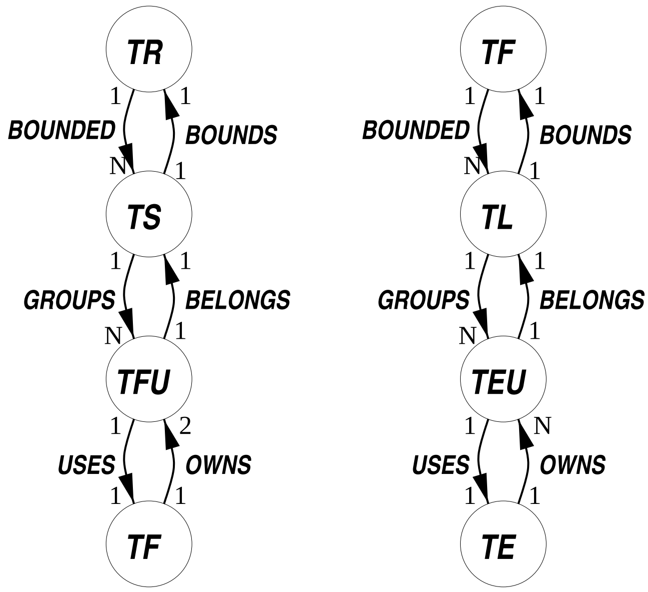


Figure 6: Analogy between the groups TR, TS, TFU, TF and TF, TL, TEU, TE.

do not store the radial adjacency for edge uses, but rather compute it on the fly.

The ACIS topological model [16] defines a TS as a collection of connected references to faces and edges. This makes it necessary to introduce additional concepts such as sidedness and containment in order to describe adjacencies of faces to solid regions. In contrast, our definition of a TS is an analogy to the bounding entity of lower dimension, the loop. Therefore, no auxiliary concepts are required in our model to specify the bounding/bounded relationship between faces and regions.

While it is possible to convert between our topological model and that of Weiler in both directions, and similarly for the ACIS model, we feel that our model is more compact and promotes insight by virtue of the direct analogy between bounding/bounded entities in the 2- and 3-manifold setting as outlined in the preceding section. The algorithms in the rest of the paper use our topological model, but could have been equally well formulated in the above alternative models, albeit at the cost of some complications.

Our topological entities are somewhat similar to Selective Geometric Complexes [17]. In particular, TF's need not be homeomorphic to 2-balls.

Acronym	Meaning
BRep	boundary representation
MV	mesh vertex
ME	mesh edge
MF	mesh face
TV	topological vertex
TE	topological edge
TEU	use of TE
TL	topological loop
TF	topological face
TFU	use of TF
TS	topological shell
TR	topological region

Table 1: Acronyms of surface mesh and BRep topological entities.

2 Overall algorithm

The main purpose of this paper is to describe an algorithm which extracts a boundary representation from oriented surface triangulations. However, in order to make our software library more flexible, we accommodate also input of polygonal models, volume (tetrahedral, hexahedral) triangulations, and raw surface triangulations by directing input through two pre-processing stages. The overall flow of data through our software library is given in Figure 7.

The most unstructured input is the *polygonal surface model*, which consists simply of polygons given through the coordinates of their vertices. In order to obtain the raw triangulation, our pre-processing algorithm deduces the incidences by merging vertices of the geometric polygons, and then triangulating the polygons. Evidently, this involves computations with tolerances, and depending on the magnitude of the tolerances, errors may be introduced either by merging vertices which should be separate or by not merging points which in fact represent a single vertex. This is the best which can be done in such cases, however, given the lack of explicit connectivity information.

Volume triangulation (tetrahedral or hexahedral meshes possibly with attached surface parts) supplies essentially the complete information for build-

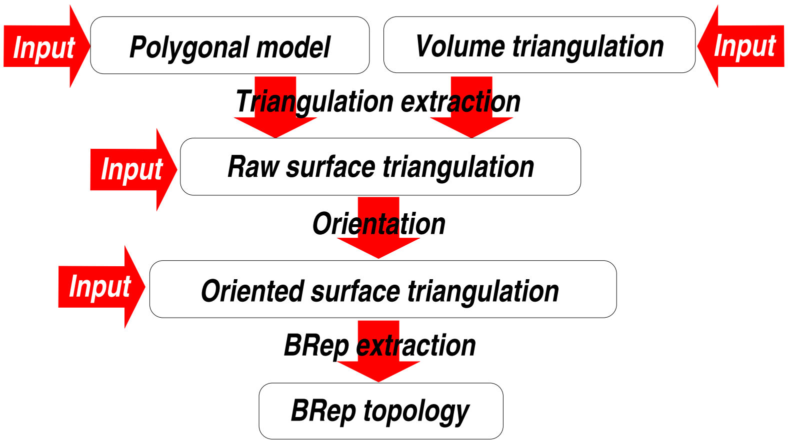


Figure 7: Overall flow of data.

ing another triangulation of the solid (i.e. for remeshing). The pre-processing stage extracts the surfaces by looking at the number of finite elements sharing a given mesh face. If there is only a single element attached to the face, or if the face is shared by two elements belonging to different solid regions, the face is marked as being part of the boundary surface triangulation.

Both the polygonal geometric model and the volume triangulation yield a raw (un-oriented) surface triangulation, which needs to be oriented into coherent pieces of surfaces for the next stage in our algorithm, extraction of the BRep. The re-orientation algorithm is discussed in the next section.

2.1 Note on Euler operators

Weiler's seminal work [10] provides a minimal list of five non-manifold Euler operators with which one can construct the BRep of an arbitrary solid given the specification of its boundary. In particular, these operators could be applied to our problem: Construct BRep based on a given triangulation. Thus, one could convert each surface mesh entity into a corresponding BRep entity using the Euler operators. Then, once the complete BRep was built, the topology simplification process would be started, using again Euler operators and some geometrical criteria, to glue together topological faces into larger ones corresponding to the smoothness of the geometry. (The analogy of this operation with the virtual topology for meshing proposed by Sheffer et al. [18] should be noted.)

Efficiency of the approach based on the basic set of Euler operators is questionable. For an input mesh of N triangles (N could be potentially very large), each of the $O(N)$ Euler operators would perform certain modification to the BRep which would be in $O(N)$ cases again undone by the action of another operator. An alternate set of richer operators could conceivably devised to make this process more efficient, but our approach follows from the question: "Is there an algorithm for building a BRep efficiently which is not based on the Euler operators?" In what follows we present one of the (possibly many) algorithms which meets this goal.

3 Orientation of raw triangulations

To orient connected pieces of the input raw surface triangulation, we pass them to the algorithm **ORIENT_TRIANGULATION**. An arbitrary MF is selected to seed the algorithm, which then proceeds to "flood" the surface until a sheet or non-manifold ME is reached. As a by-product, the surface underlying the triangulation is marked non-orientable if any neighbor of the currently considered triangle is tagged as "oriented" and the orientation of the two triangles is incompatible. In this case, the triangulation t is not *oriented* in the sense the definition from Section 1 (the solution is not optimal), but can be used as is as far as its applicability in the algorithms of this paper goes.

```
algorithm ORIENT_TRIANGULATION (Triangulation t)
  Tag all triangles as ''not oriented'';
  Stack S push [an arbitrary MF from t];
  while (S not empty) do
    mf ← pop S;
    Tag mf as ''oriented'';
    foreach (ME me of mf) do
      if (me is manifold) then
        omf ← MF across me;
        if (omf not oriented) then
          Orient omf compatibly with mf;
          Stack S push omf;
        else
          if (Orientations of omf and mf incompatible) then
            Mark surface underlying t as not orientable;
          endif
        endif
      endif
    done foreach
  done while
```

It is interesting to note the effect use of the **ORIENT_TRIANGULATION** algorithm has on non-orientable surfaces; see Section 10.

4 Extraction of faces

The topological faces are identified first. The underlying geometrical representation of each TF is a collection of mesh faces which are (uniquely) classified on the TF. The first step in our algorithm is therefore to group the MF's into representations of distinct TF's. While any MF is still not classified on a TF, the algorithm **MAKE_TF** is applied. It uses a stack data structure to "flood" the surface by starting out from a single MF, then passing on to its neighbors and so on. The neighbor of a MF is pushed to the stack only if

1. the ME separating them is manifold (exactly two triangles are joined along the edge);

2. the neighbor is not classified on any TF yet;

3. the neighbor is oriented compatibly; (This clause takes care of non-orientable surfaces.)

4. the ME does not correspond to a crease in the surface.

The decision if an ME lies on a crease is purely geometrical. One possibility is to use the geometry of the discrete surface, another option is to build some smooth surface on top of the geometry of the discrete surface, and then apply mesh optimization based on some error criteria. The latter approach has been advocated by Hoppe et al. [19], who used subdivision surfaces with varying smoothness.

Each TF keeps track of the MF's classified on it, and also of ME's which bound the TF. The ME's which the algorithm recognized as bounding the current TF are marked as a by-product during the collection of the MF's in **MAKE_TF**.

```
algorithm MAKE_TF (MF mf)
  tf ← new TF;
  Stack S push mf;
  while (S not empty) do
    mf ← pop S;
    if (mf not classified on any TF) then Classify mf on tf; endif
    foreach (ME me of mf) do
      if (me is manifold) then
        omf ← MF across me;
        if (omf not classified on any TF
            and omf and mf are oriented compatibly
            and me does not correspond to a crease) then
            Stack S push omf;
        else if (omf not classified on tf) then
            Classify me as bounding tf;
        endif
      else {either sheet or non-manifold edge }
        Classify me as bounding tf;
      endif
    done foreach
  done while
```

5 Extraction of edges

TE's are detected next. Each TF is examined in turn, and while there is any ME marked as bounding the TF which has not been classified on any TE yet, the algorithm **MAKE_TE** is run. It is a one-dimensional analog of **MAKE_TF** in that the neighboring ME joined to the current ME at one of its vertices is pushed into the stack if

1. there are exactly two ME's joined at the vertex;
2. the vertex does not correspond to a "corner."

The recognition of a corner in the TE sequence is again, analogously to the case of creases in surfaces, purely geometrical decision. For the sake of simplicity, we use the turning angle as our "corner" criterion.

```
algorithm MAKE_TE (ME me)
  te ← new TE
  Stack S push me;
  while (S not empty) do
      Classify [me ← pop S ] on te;
      foreach (MV mv of me) do
          n ← count ME's incident to mv (except me);
          if (n equal 0) then
              Bound te by [tv ← (new) TV at mv];
          else if (n equal 1) then
              ome ← other ME at mv than me;
              if (ome not classified on any TE) then
                  if (me and ome do not correspond to a corner)
                  then Stack S push ome;
                  else  Bound te by [tv ← (new) TV at mv]; endif
              else
                  if (ome classified on te and te is a cycle)
                  then Bound te by [tv ← TV at mv];
                  else  Bound te by [tv ← (new) TV at mv]; endif
              endif
          else {Branching of TE's }
              Bound te by [tv ← (new) TV at mv];
          endif
      done foreach
  done while
```

As a by-product of the TE classification is that topological vertices are identified as "ends" of TE's. Note that TV's are created at MV's just once, and are then re-used as needed (hence the parentheses around "new"). If a TE is not bounded by any TV's naturally (i.e. if it is a cycle), a TV is created at any MV on the TE, and the TE is updated to start and end at this TV.

Finally, the ME's along the TE are ordered in sequence from the first MV on the TE to its last. For a cyclic TE, the ordering (and the orientation of the TE) are a matter of arbitrary choice. The ordering of ME's provides an unambiguous definition of geometry (and orientation) even for cycle edges.

6 Extraction of loops

Next, topological loops are extracted for each TF. All loops are on the same footing in our topological model, i.e. no distinction is made between loops corresponding to holes in surfaces and external boundaries of surfaces. Such distinctions are based on *geometry*, and we choose not to inject them into the topology.

To make certain operations unambiguous and simple, we elect to link the orientation of the surface with that of its topology. The loops are oriented in such a way as to comply with the following rule: When looking against the normal to the TF (which is the same as that of the underlying MF's), and along the loop, the TF is to the left of the loop. (In other words, orient your right hand with the surface on the side of the palm of your hand, the thumb in the direction of the normal, and your fingers point along the loop.)

While there is any TE *ste* bounding the TF *tf*, which has not been used in any TL on *tf* yet, the algorithm **MAKE_TL** collects another loop, seeding the algorithm with *ste*. We start by collecting all TE's constituting a connected chain with *ste*. Then, for all TV's on these edges we collect those TE's that can be used to *leave* given TV while tracing the boundary of the face. Figure 8 illustrates this algorithm. Consider TV 1 as our current viewpoint. Given our link between the loop topology and the geometric orientation of the surface as explained above, we can see that to leave TV 1, one can use only those TE's along mesh edges which point out of TV 1, i.e. TE 1 and TE 3. TEU's of both edges are at this point added to the loop being built.

```
algorithm MAKE_TL (TF tf, TE ste)
  tl ← new TL on tf;
  tvl ← List of TV's on TE's on tf connected to ste;
  foreach (tv in tvl) do
    tel ← List of TE's leaving tv;
    foreach (te in tel) do
      if (tv equal first TV of te) then Orientation(te) ← SAME;
      else                              Orientation(te) ← REVERSED;
      endif
      Add [teu ← new TEU of te with Orientation(te)] to tl;
    done foreach
  done foreach
```

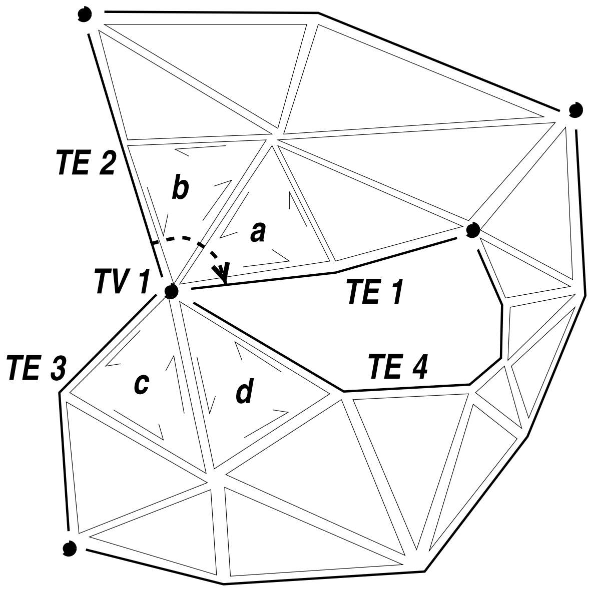


Figure 8: How to decide which TE leaves TV 1.

The algorithm **MAKE_TL** can be compared with that of Wilson [20] based on graph traversals. Our algorithm is an improvement in that it can handle multiple uses of edges, such as presented in Figure 3; compare also with Reference 9.

7 Extraction of shells

Recovery of shells as boundaries of solid regions is the most difficult part of our algorithm. To understand why, we need to consider the origin of the input surface meshes.

7.1 Surface-volume adjacencies are specified

Surface meshes can be extracted from volume triangulations, in which case the surface-volume adjacencies are fully specified. The regions on each side of the surface triangulation (i.e. on each side of each TF) can be queried, and regions and shells can be established easily.

As long as there is any TF bounding a not-yet-extracted TR, the algorithm **MAKE_TR** is applied. All the shells bounding the TR are collected as a side-effect.

```
algorithm MAKE_TR
    tr ← new TR;
    while (There is any TF tf bounding tr which is not
            used by any TS bounding TR yet) do
        perform MAKE_TS (tf, tr);
    done while
```

```
algorithm MAKE_TS (TF tf, TR tr)
    ts ← new TS;
    Stack S push tf;
    while (S not empty) do
        tf ← pop S
        if (tf has tr on the SAME side?) then
            Add [tfu ← new TFU of tf with Orientation REVERSED] to ts;
        endif
        if (tf has tr on the REVERSED side?) then
            Add [tfu ← new TFU of tf with Orientation SAME] to ts;
        endif
        foreach (TL tl on tf) do
            foreach (TEU teu in tl) do
                TE te ← TE of teu;
                foreach (TEU oteu of te) do
                    TF otf ← TF on which oteu used in a TL;
                    if (otf not equal tf and otf not used by ts) then
                        Stack S push otf;
                    endif
                done foreach
            done foreach
        done foreach
    done while
```

7.2 Surface-volume adjacencies not available

Surface meshes coming from unconnected geometric (e.g. polygonal) models, and facetted or triangulated CAD surfaces, come (typically) without any information about which volume is bounded by which surfaces. Therefore, the best we can do in these cases is to make some guesses. First of all, we restrict ourselves to bounded regions.

The overall algorithm of shell and region extraction is described next.

1. Gather groups of connected TF's. (This is easily accomplished via edge-face adjacencies.) Each such group corresponds to one or more shells.

2. Determine for each group if it is manifold, sheet, or non-manifold, and create all possible shells, open and closed. While this step is easy for sheet and manifold shells, it is more complicated for non-manifold shells as discussed below.

3. For each shell *ts*, determine the shells which are immediately contained by *ts*, and the shell which immediately contains *ts*.

4. Create regions. Group regions enclosed by shells on the same level of the containment tree, if desired.

7.2.1 Shells

The TFU's constituting a shell are collected by using the radial ordering of faces around topological edges [10]. Given our underlying geometric representation of TF's by triangulations, we can easily establish cyclic lists of faces around an edge by the algorithm **RADIAL_ORDER** (compare also with Figure 9).

```
algorithm RADIAL_ORDER (TE te): Cyclic list of MF's
  me ← first ME classified on te; {ME's are ordered along te }
  Cyclic List L ← all MF's joined along me;
  firstmf ← arbitrary MF in L;
  foreach (mf in L) do
     Compute positive, oriented angle α through which one needs
        to rotate firstmf around me into the plane of mf (positive α
        turns counterclockwise when looking against the tangent to te;
        see Figure 9);
  done foreach
  Order L based on the magnitude of α;
  return L;
```

Weiler's thesis proposes an algorithm for determination if a set of faces encloses a three-dimensional region [10]. Our algorithm is slightly simpler, and the differences are mostly due to the data structures used. The same algorithm can be used for gathering TFU's for closed manifold or non-manifold shells.

```
algorithm MAKE_TS_G (TF tf, Orientation(tf))
  ts ← new TS;
  Add [tfu ← new TFU with Orientation(tf)] to ts;
  Stack S push tfu;
  while (S not empty) do
    tfu ← Pop S;
    tf ← TF used by tfu;
    foreach (TL tl on tf) do
      foreach (TEU teu in tl) do
        te ← TE used by teu;
        mfradl ← RADIAL_ORDER (te);
        foreach (MF mf in List mfradl) do
          if (mf located on tf) then
            otf ← immediate radial neighbor of mf;
            otfo ← Orientation compatible along te with tfu;
            if (otf not used in ts with Orientation otfo) then
              Add [tfu ← new TFU with Orientation otfo] to ts;
              Stack S push tfu;
            endif
          endif
        done foreach {MF loop }
      done foreach {TEU loop }
    done foreach {TL loop }
  done while
```

It is noteworthy that the algorithm **MAKE_TS_G** for the construction of shells is not quite as simple as that for the loops (**MAKE_TL**), despite the direct analogy topological entities for loops (bounding faces) and shells (bounding regions). The reason is that while for the loops we know which TE's bound given TF, we do not know the equivalent relationship for the shells. Furthermore, the direction of traversal is well defined for a loop, but not so for a shell.

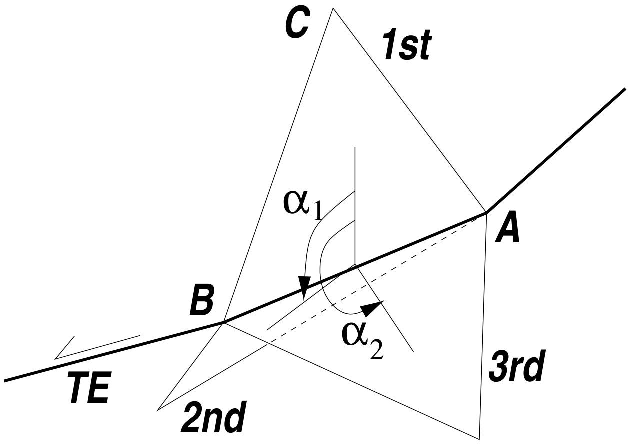


Figure 9: Radial ordering of faces around an edge.

7.2.2 Containment tree

Containment decisions are based on geometric tests, namely on the number of intersections of a ray fired from the surface of the tested shell with other shells. For the purpose of the containment analysis we distinguish between two types of shells, shells bounding an unbounded three-dimensional pointset (type U), and shells bounding a bounded three-dimensional pointset (type B). (Note that sheet shells are of type U.) An important property of the containment tree is that (closed) shells of type U can contain only shells of type B and vice versa. The root of the containment tree is always a TS of type U (it bounds the "infinite" complement of a bounded 3-D solid). Figure 10 illustrates these concepts. Each TF is drawn as two parallel lines corresponding to its two TFU's (the figure portrays each TFU as a line in a plane, but the concepts translate directly to three dimensions). The orientation of each TS is shown as an arrow anchored on the shell. The shell numbers are subscripted with their type.

The tree is built by determining for each shell which TS's are contained in it. All shells which are not contained in any other shell become roots of containment trees. (We adopt the terminology of Reference 21, but not its algorithms.) Entering each tree at the root, we determine immediate descendants of each leaf. (Immediate descendant of *ts* are all shells which are contained in *ts* but are not contained in any other shell on lower level.)

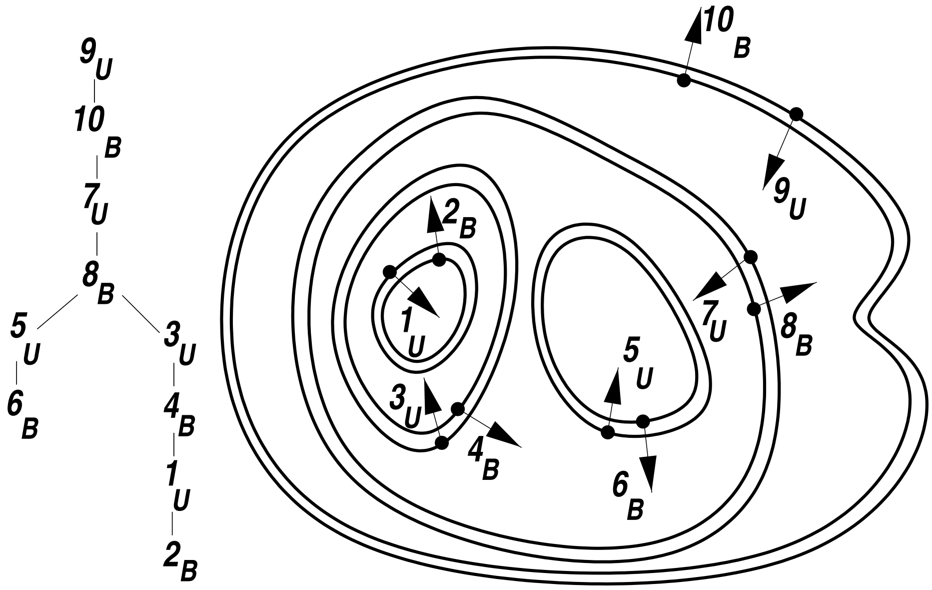


Figure 10: Containment of shells.

7.2.3 Creation of regions

At this point we can create regions based on the containment tree. Every shell of type B may enclose a bounded 3-D pointset, with all its immediate descendants (recall that they are of type U) representing boundaries of voids in the pointset. Regions on the same level in the containment tree can be finally grouped, because in many cases those regions are of the same material. (Consider, for example re-enforcing bars in a concrete beam, or voids in a foam). More complex strategies are also possible as demonstrated by Cavalcanti et al [22].

The left part of Figure 10 shows the containment tree, from which it is possible to derive, for instance, the regions shown schematically in Figure 11.

8 Geometry redefinition

Once the BRep has been built, one can modify (improve) the definition of the geometry description of the faces and edges. The technique that seems of particular value are subdivision surfaces [23]. For instance, piecewise smooth surface reconstruction based on mesh optimization has been proposed by

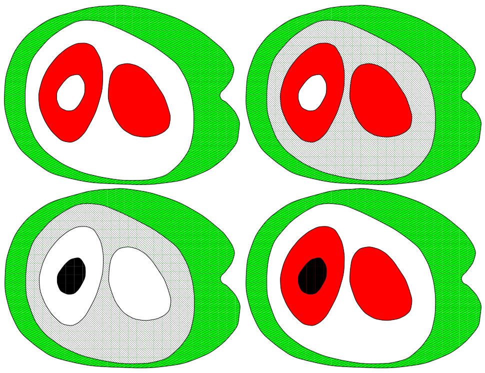


Figure 11: Some region definitions based on the containment tree of Figure 10.

Hoppe et al. [19]. Applying the concept of piecewise smooth subdivision surfaces to our geometry/ topology model is rather straightforward, and what seems to be of even greater value is using these mesh optimization techniques to detect the non-smooth features (creases, corners) on which topological entities should be located. This question is currently being investigated.

9 Model verification

The finished BRep should be verified in a number of ways. Among the tools available for this purpose are the Betti numbers [15, 24]. Euler characteristic is a particularly valuable invariant; Reference 25 gives a useful background information (although it draws wrong conclusions from it).

In this work, we choose to apply the model verification at the level of topological shells. As each TS is assembled either by the **MAKE_TS** or **MAKE_TS_G** algorithm, a number of checks are performed. The topological checks are evaluated on the simplicial subcomplex classified on the TFU's grouped in the TS, not on the BRep topological entities themselves.

Number of components The number of unconnected components corresponds to the zeroth Betti number. Since a TS is by definition connected, the number of components is one.

Orientability Orientability can be checked for sheet or manifold shells through the by-product of the algorithm **ORIENT_TRIANGULATION**. For non-manifold shells, non-orientable components (such as embedded Möbius bands, or Klein bottles) are not necessarily detected by **ORIENT_TRIANGULATION**, and should be checked either by computing directly the corresponding Betti number, or by running a mesh self-intersection test (for example to detect closed, but non-orientable shells).

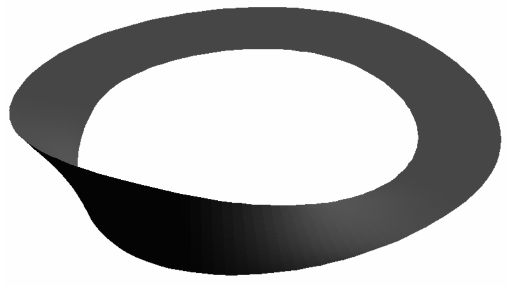
Closedness To check if a TS is closed or not is particularly simple. Examine all TF's used by the TS. If any TF is used just once by the TS, the shell is closed (i.e., it can enclose a three-dimensional volume, provided it is orientable).

Genus The number of handles can be computed from the Euler characteristic of the subcomplex associated with the shell. The Euler characteristic is computed as $\chi = \#MV - \#ME + \#MF$, where $\#MF$ is the number of MF's classified on one of the TF's used by the TS, and $\#ME$ and $\#MV$ are the totals of ME's and MV's shared by the MF's. Genus g is computed from the Euler characteristic as $g = 1 - \chi$ (open shells) or $g = 1 - \chi/2$ (closed shells).

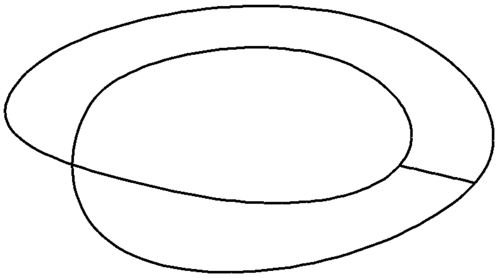
10 Examples

10.1 Non-orientable surfaces

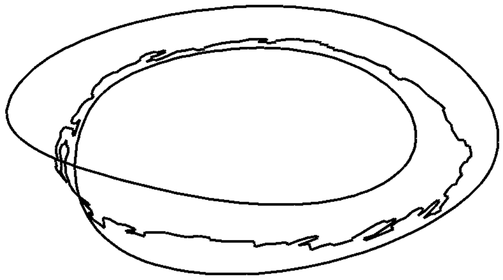
The Möbius strip is considered here as a prototypical non-orientable surface. The input is a polygonal model, which is converted to raw triangulation. The orientation algorithm of Section 3 is then run to produce an oriented triangulation. In the first instance, the triangles are input in a regular fashion, and the **ORIENT_TRIANGULATION** algorithm detects a "forced" TE due to the non-orientability of the surface, which cuts the ring into a simply connected face (Figure 10.1b). In the second instance, the triangles are input in random order, and the orientation algorithm constructs a TE which cuts the strip "parallel" to its boundary. The result is then a cylindrical face with two twists (Figure 10.1c).



(a) Shaded view of the surface

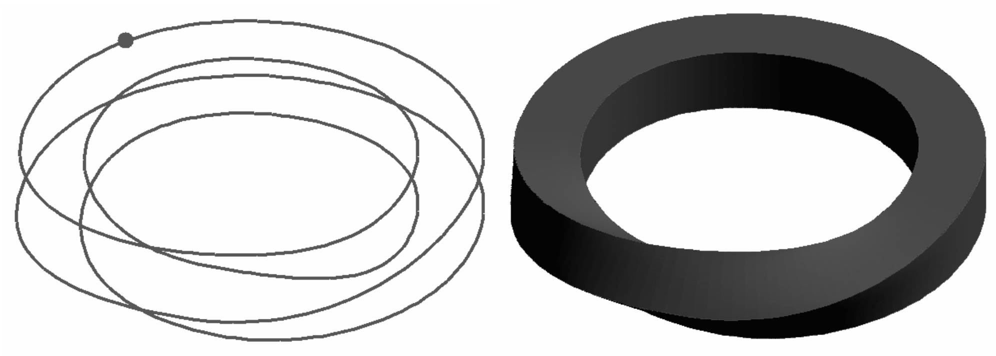


(b) TE's for regular input sequence



(c) TE's for randomized input sequence

Figure 12: The Möbius strip. BRep for two input sequences.



(a) Topological edges. (b) Shaded view.

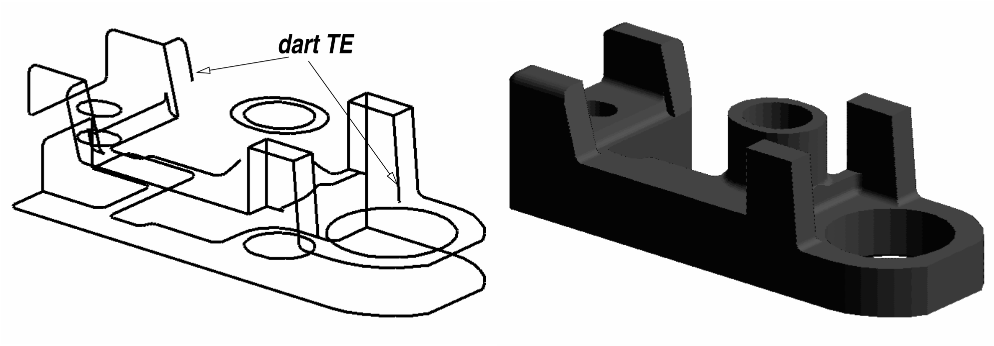
Figure 13: BRep of the twisted ring.

10.2 Twisted ring

Due to the complete freedom in the construction, our topological models are not complexes for which the Euler characteristic makes sense. Consider, for example, the twisted ring, whose physical model is obtained by cutting a ring with a rectangular cross-section, giving one end a 90 degree twist, and re-gluing the cut ends together. Our algorithm creates a topological model with one TV, one TE, and one TF (compare with Figure 13). Evidently, the Euler characteristic computed from the topological model cannot be correct (1TV - 1TE + 1TF $= 1 \neq 0$), and the reason lies in the genus of the TF, which is not homeomorphic to a 2-disk (eventhough it has only one boundary TL!). In fact, the surface is a four times twisted strip. However, because of the possibility of evaluating the Euler characteristic on the mesh underlying the topological entities, the topological invariants can be verified to yield correct answers (one component, orientable surface with one handle).

10.3 Gehäuse

Figure 14 shows the BRep of a quite well-known CAD part, the MBB "gehäuse" [26]. There is a number of fillets and rounded corners in this model, yielding a large number of faces when their boundaries are based on type. The present approach, which is based on the concept of "natural" surface boundaries based on smoothness makes some faces rather large in ex-



(a) Topological edges. (b) Shaded view of topological faces.

Figure 14: BRep of “Gehäuse.”

tent; compare with Figure 15. (Arrows indicate so-called dart edges, which correspond in this case to intersection curves of two surfaces.)

The surface mesh (1780 triangles, 2681 edges) orientation took 0.03 seconds, and BRep was built in 0.11 seconds.[1]

Figure 15 shows one of the TF's. Note that the algorithm gathered into this face all “smoothly” connected surfaces (in this case, those which are joined in a G^1 fashion).

10.4 Torpedo motor

The geometry of the torpedo motor has been defined as a facetting of the ACIS geometry available on the National Design Repository Web server [27]. The input file to our algorithm consisted of 85,542 triangles. Topology of the torpedo motor has been extracted in 4.7 seconds. Summary of the topological model is provided in the following box. Note, in particular, the high genus of the object (71). (The Euler characteristic is computed from the underlying mesh, not from the topological entities.)

[1] All timings in this paper given as wall-clock measurements on an 225 MHz SGI Octane workstation with R10K processor, 128 MB of memory, and 32 KB instruction and data caches, and 1 MB secondary cache.

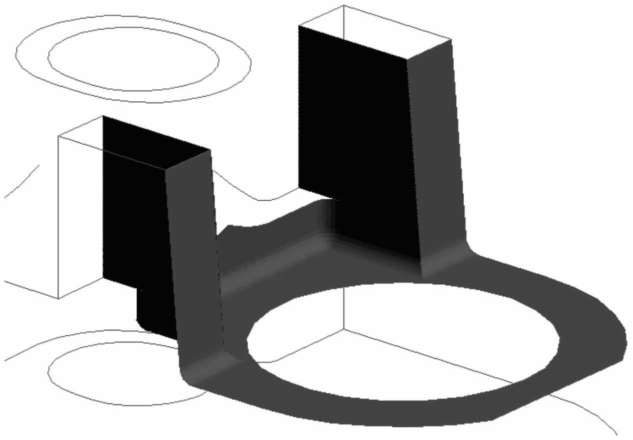


Figure 15: BRep of “Gehäuse.” Detail showing one of the topological faces.

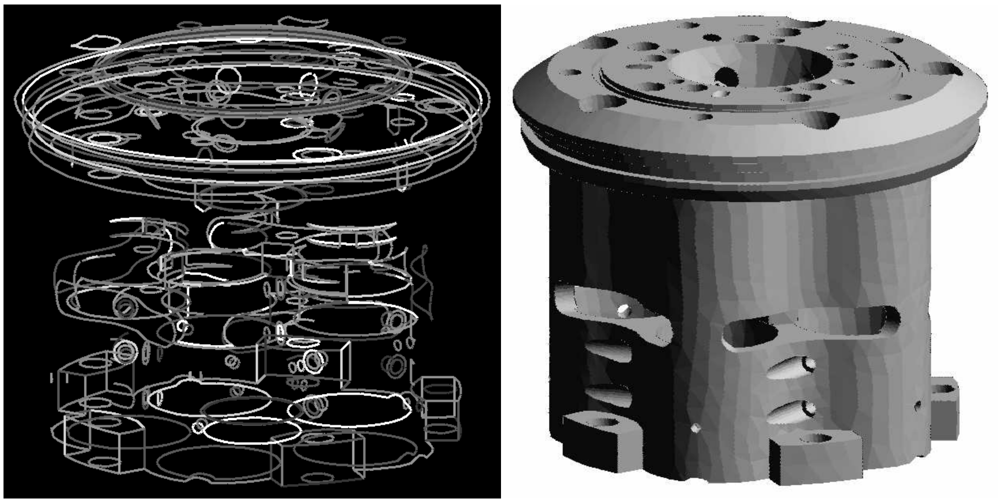
```
TOPOLOGICAL DATA STRUCTURE for TORPEDO MOTOR:

Totals: #tv 569, #te 635, #tl 528, #tf 231, #ts 1, #tr 1
Limits: crease angle = 60, corner angle = 45
Mesh is orientable


...
TS 1
TS is closed.
Euler characteristic, chi: -140
Genus [= (2 - chi) / 2]: 71
TF: ( 1 44 11 13 ... )
...
```

10.5 Multi-material solid

In this example we demonstrate use of our algorithm in a non-manifold setting. The geometry is composed of three blocks (the two larger blocks contain holes to allow for the passage of the smaller blocks). The input surface mesh (10,150 triangles, 15,158 edges) was extracted from a tetrahedral triangulation of the domain (9,024 nodes, 46,747 tetrahedra), in 0.41 seconds, and the



(a) Topological edges. (b) Shaded view.

Figure 16: BRep of torpedo motor.

BRep was built in 0.53 seconds. Figure 17 shows the input triangulation (the outermost region is partially clipped to allow a view inside the domain).

As the timings indicate, the BRep for this size of models can be built under one second of real time, which compares favourably with the current volume triangulation times (approximately 1,000 to 10,000 tetrahedra per second). This makes our technique interesting in simulations of changing topology (for example fracture, fragmentation, or wetting).

Conclusions

A methodology for extraction of boundary representation topological models of solids from surface triangulations has been presented in considerable detail, including pseudo-code of the main algorithms.

Our approach is not based on Euler operators, mainly for efficiency reasons. The technique we describe consists of a number of steps in which we identify topological faces, edges, loops and finally shells and regions in that order. Topological information is supplemented by geometrical estimators of surface and curve smoothness. Although we use only the simplest criteria,

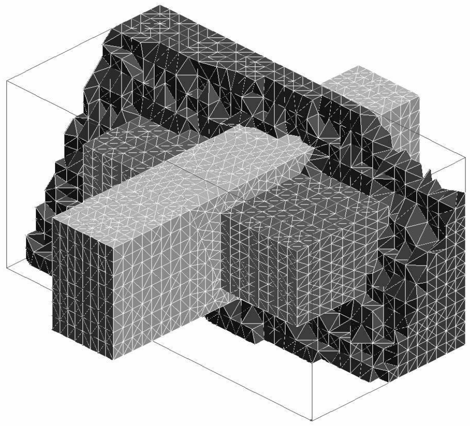


Figure 17: Multiple-region, non-manifold model with tetrahedral mesh. (The outermost region is partially clipped off.)

namely the dihedral and turning angle, more complicated procedures able to handle noisy inputs are currently being investigated.

The BRep extraction procedure has been shown to be robust and fast on a number of examples, and it is felt that our current efforts to incorporate subdivision surface algorithms and processing of noisy surface triangulations will make this an even more useful computational tool.

Acknowledgements

We are grateful for support from the Department of Energy through Caltech's ASCI Center of Excellence for Simulating Dynamic Response of Materials.

References

[1] M. S. Shephard and M. K. Georges. Reliability of automatic 3D mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 101(1–3):443–462, 1992.

[2] P. M. Finnigan, A. Kela, and J. E. Davis. Geometry as a basis for finite element automation. *Engineering with Computers*, 5:147–160, 1989.

[3] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering*, 40:1573–1596, 1997.

[4] A. A. G. Requicha and J. R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, September:31–44, 1992.

[5] A. Pandolfi and M. Ortiz. Solid modeling aspects of three-dimensional fragmentation. *Engineering with Computers*, 14(4):287–308, 1998.

[6] A. A. G. Requicha and H. B. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 23(1):30–44, 1985.

[7] E. L. Gursoz, Y. Choi, and F. B. Prinz. Boolean set operations on non-manifold boundary representation objects. *Computer-Aided Design*, 23(1):33–39, 1991.

[8] C. M. Hoffmann and J. R. Rossignac. A road map to solid modeling. *IEEE Trans. Visualization and Computer Graphics*, 2(1):3–10, 1996.

[9] J. Rossignac and D. Cardoze. Matchmaker: Manifold BReps for non-manifold r-sets. Technical Report SM99-020, Georgia Institute of Technology, 1999.

[10] K. Weiler. *Topological structures for geometric modeling.* PhD thesis, Rensselaer Polytechnic Institute, 1986.

[11] S. H. Lo. Volume discretization into tetrahedra-I. Verification and orientation of boundary surfaces. *Computers and Structures*, 39(5):493–500, 1991.

[12] S. H. Lo. Analysis and verification of the boundary surfaces of solid objects. *Engineering with Computers*, 14:36–47, 1998.

[13] R. Löhner. Regridding surface triangulations. *J. Computational Physics*, 126:1–10, 1996.

[14] P. N. Chivate and A. G. Jablokow. Solid model generation from measured point data. *Computer-Aided Design*, 25(9):587–600, 1993.

[15] L. Ch. Kinsey. *Topology of surfaces.* Springer Verlag, 1993.

[16] Spatial Technology Inc., Boulder, CO. *ACIS 3D Toolkit 3.0 Modeling Primer*, 1997.

[17] J. R. Rossignac and M. A. O'Connor. Sgc: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wozny, J. Turner, and K. Preiss, editors, *Geometric modeling for product engineering*, pages 145–180. North-Holland, 1989.

[18] A. Sheffer, T. Blacker, and M. Bercovier. Virtual topology operators for meshing. In *Proc. 6th International Meshing Roundtable*, pages 49–66, 1997.

[19] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jina, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 295–302, 1994.

[20] P. R. Wilson. Euler formulas and geometric modeling. *IEEE Computer Graphics and Applications*, pages 24–36, 1985.

[21] I. Gargantini, G. Schrack, and A. Kwok. Reconstructing multishell solids from voxel-based contours. *Computer-Aided Design*, 26(4):293–301, 1994.

[22] P.R. Cavalcanti, P.C.P. Carvalho, and L.F. Martha. Non manifold modelling: An approach based on spatial subdivision. *COMPUTER-AIDED DESIGN*, 29(3):209–220, 1997.

[23] P. Schröder and D. Zorin. Subdivision for modeling and animation. In *SIGGRAPH 98 Course Notes*, 1999.

[24] C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12:771–784, 1995.

[25] A. G. Jablokow, Jr. J. J. Uicker, and D. A. Turcic. Topological and geometric consistency in boundary representations of solid modles of mechanical components. *Trans. of the ASME*, 115:762–769, 1993.

[26] R. Hillyard. The Build group of solid modelers. *IEEE Computer Graphics and Applications*, March:43–52, 1982.

[27] The NIST Design, Process Planning and Assembly Repository. http://edge.mcs.drexel.edu/repository, 1999.