Topological Construction of 2-Manifold Meshes from Arbitrary Polygonal Data

VINOD SRINIVASAN and ERGUN AKLEMAN *
Visualization Sciences Program
College of Architecture

JOHN KEYSER †
Department of Computer Science
College of Engineering

Texas A&M University

Abstract

In this paper we present a simple algorithm to construct 2-manifold meshes from arbitrary collections of polygons. We form our final data structure using two very basic manifold-preserving operations, thus guaranteeing that the result is a valid manifold. Our algorithm is purely topological and does not consider the geometric properties of the underlying shape.

The algorithm automatically and correctly creates the missing faces of manifolds with boundaries. It also eliminates all twogons (2-sided polygons) and converts non-manifold meshes into one of the possible manifold interpretations. We have implemented this algorithm and we highlight the performance of our algorithm on a number of sample models.

1 Introduction and Motivation

When modeling solid objects, the construction of a manifold representation is often desirable, although it can be difficult to achieve. In this paper, we present an algorithm for constructing manifolds from a given collection of arbitrary polygonal data.

1.1 Motivation

The transfer of data between various systems and file formats can be the cause of significant problems when modeling solids. Often the output to an intermediate file format loses much of the topological information that a particular system maintains internally. In other cases, data that is valid in one system creates problems in another.

Graphics file formats in particular often omit topological information, as only the geometric definitions of polygons are necessary for display purposes. For example, Alias-Wavefront's obj file format is one of the more common graphics file formats, and geometry is fundamentally stored as just a collection of oriented polygons.

For many solid modeling and shape modeling applications, we desire that the meshes describing object boundaries be represented as valid and correct 2-manifolds in 3D. Certain algorithms (e.g. certain subdivision surface and object intersection algorithms) rely on this property to guarantee correctness. This issue becomes apparent in very practical situations. For instance, any modeling system fails when Doo-Sabin subdivision [9, 17] is applied to a non-manifold mesh since Doo-Sabin requires manifold meshes [3].

We would like to have a simple and straightforward way to construct valid manifold meshes from an arbitrary set of polygons. Such a method will allow the importation and construction of objects from files that lack direct topological connectivity, and will allow the exploration of a wide variety of manifold solids. Furthermore, we would like to use only topological operations to perform this construction. While geometric considerations have great usefulness and may address many of the problems encountered, the utility of such techniques will vary depending on the particular application. Since such geometric techniques are not universally applicable, and will need to be "tuned" to particular data sets, we choose to avoid them.

1.2 Main Results

We present an algorithm for constructing manifold solids from an arbitrary collection of oriented polygons. This algorithm reads in a collection of oriented polygonal faces, each described by an ordered sequence of vertices. Using only topological operations, it constructs a manifold solid from this data. The algorithm has the following properties:

- The output is always a manifold. If the original data corresponds to a single valid manifold solid, then that is the manifold that will be output.

* Address: 216 Langford Center, College Station, Texas 77843-3137. email: ergun@viz.tamu.edu. phone: +(979) 845-6599. Supported in part by the Texas A&M, Scholarly & Creative Activities Program.

† Address: Department of Computer Science, College Station, TX 77843-3112. email: keyser@cs.tamu.edu. Supported in part by NSF Award 0138446

- Each face is oriented in a way consistent with the other faces in the manifold.
- There are no twogons (two-sided faces).
- No new vertices are created.
- No looping edges (edges from one vertex to the same vertex) are created.

There are three significant results in the work we present here. These are:

1. *Crack repair.* Our algorithm provides a simple method to automatically fill holes in the input model. That is, if the input polygons correspond to a manifold with boundary, the output will be a manifold. We do not need to explicitly determine the boundaries of holes. Two examples of manifolds with boundaries are shown in Figure 1.

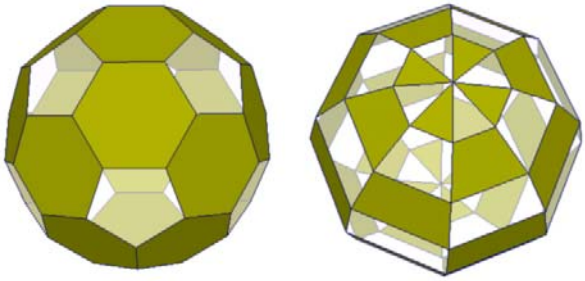

Figure 1. Examples of manifolds with boundaries.

2. *Generating manifolds from non-manifolds.* Given input that corresponds to a non-manifold (including cases such as dangling faces), we generate a manifold interpretation of the data. Note that there might be many possible manifold interpretations for the given data as shown in Figure 2. We guarantee only that the resulting structure fits one of those interpretations.
3. *Insight into manifolds.* By intentionally introducing artifacts into a data set such as wrongly-oriented polygons or non-manifold situations, we can create a number of strange (but still manifold) objects. Our approach allows us the freedom to explore a wide variety of manifold object representations that are not generally created or considered when modeling solids.

1.3 Geometric Considerations

Note that because our algorithm is based entirely on topology, it totally ignores the geometric properties of the underlying shape. For instance, the geometry might be self-intersecting, vertices might be coincident, and vertices of polygonal faces might not be coplanar. Such can hold true for both the input and the output of our algorithm. Similar problems may result from *any* purely topology-based approach.

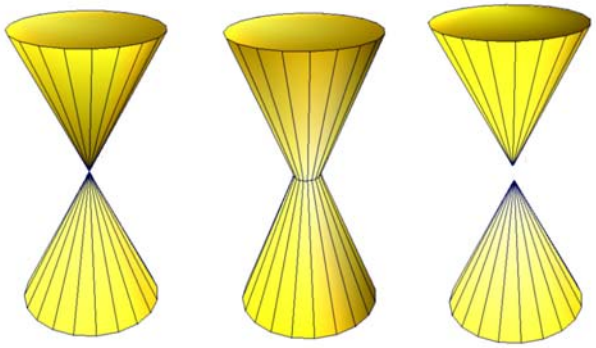

Figure 2. Two 2-manifold interpretions of a non-manifold.

While the focus of our approach is strictly topological, a number of geometric operations could be applied as a preprocess or postprocess step. For example, coincident vertices in the input could be easily merged before applying our algorithm. While such steps eliminate some geometric problems, the output may still contain geometric irregularities, given sufficiently difficult input. For example, non-manifold input, wrongly-oriented faces, or non-planar cracks can lead to output polygons for which the vertices are not coplanar. Although this may be undesirable in certain situations, we view this as a potential benefit—it allows us to examine the manifold representations possible when non-standard data (such as a "wrongly-oriented" face or non-manifold situation) is provided.

2 Previous Work

Akleman and Chen recently introduced topologically robust mesh modeling operators [1] based on topological graph theory [8, 11]. Topological graph theory is a relatively obscure mathematical theory introduced almost 100 years ago [10, 13] and familiar to only a handful of graph topologists. Akleman and Chen's INSERTEDGE and DELETEEDGE [1] operators can effectively change the topology of a manifold mesh by inserting and deleting handles. They also have recently shown [5] that all and only orientable 2-manifold structures can be created using the two simplest Euler operations, MVFS (make a surface with a single vertex and a single face) and KVFS (inverse of MVFS) [15], along with INSERTEDGE and EDGEDELETE. Moreover,

these four operators can be efficiently implemented [5] on almost every mesh data structure including winged-edge, [7], half-edge [15] and quad-edge[12]. These results suggest that software development for mesh modeling with a topologically guaranteed orientable manifold property can be greatly simplified using only these four operators.

3 Methodology

Among the four operators, two are sufficient to construct our algorithm. These are INSERTEDGE and CREATEVERTEX:

1. CREATEVERTEX(v) is similar to the Euler operator MVFS. It creates a new isolated vertex v and its corresponding face; we call this a point-sphere (see Figure 3.) This operator effectively adds a new surface component to the current 2-manifold.

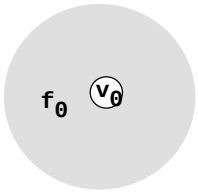

Figure 3. A point sphere is a manifold that consists of only one vertex and one face without an edge.

   The CREATEVERTEX operator is essential in the initial stage of manipulating 2-manifolds since it is needed to create and add a new surface component in an empty manifold.

2. INSERTEDGE(c_1, c_2, e) inserts a new edge e to the mesh structure between two corners c_1 and c_2. Formally, a *corner* is a subsequence of a face boundary walk consisting of two consecutive edges plus the vertex between them except in the case of a point sphere, in which the vertex itself is the corner. If a face has only one instance of each vertex, then we can consider a corner as a face-vertex pair which is given as $\{f_i, v_j\}$. We can also consider a corner as a subsequence of a face boundary walk consisting of three consecutive vertices. For example if $f = \{v_1, v_2, v_3, v_4\}$ is a face, $\{v_1, v_2, v_3\}$ is a corner referring to the vertex v_2 in face f and $\{v_3, v_4, v_1\}$ is the corner referring to the vertex v_4 in face f.

   If INSERTEDGE inserts an edge between two corners of the same face, the new edge divides the face into two faces. On the other hand, if INSERTEDGE inserts an edge between corners of two different faces, the new edge merges the two faces into one. See the figures in [1] for visual explanation.

   When the operator INSERTEDGE is inserting an edge e between corners of two different faces f_1 and f_2, there is an interesting and intuitive interpretation of the operation as two steps. In the first step, we delete the faces f_1 and f_2, which results in a 2-manifold with boundaries because of two "open" holes left by deleted faces f_1 and f_2. In the second step we run a new "pipe" between these two holes and allow the pipe ends to "seal" the two holes.

   Note that the choice of corners is extremely important for the INSERTEDGE operation. An edge inserted between two vertices may combine two faces into one or separate one face into two depending on the choice of corners.

Figures 4, 5 and 6 show how a tetrahedron can be constructed by a series of CREATEVERTEX and INSERTEDGE operations. The following section presents an algorithm that uses only the above mentioned operators to construct a 2-manifold mesh from a list of vertex positions and a list of oriented polygons given as a sequence of vertex indices.

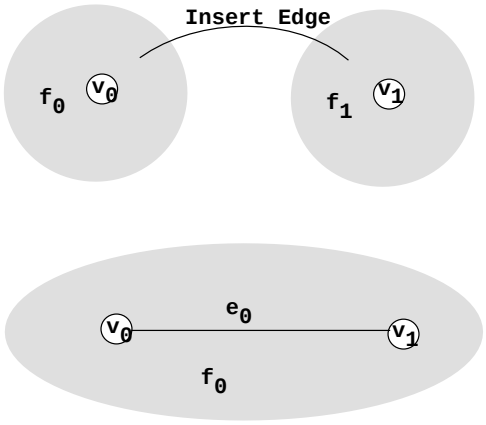

Figure 4. A line manifold can be obtained by combining two point spheres with an insert edge operation. Note that the insert edge operation combines the two initial faces f_0 and f_1 and the result has only one face.

3.1 Overview of the Algorithm

The algorithm consists of three steps.

1. Read all vertex positions and create a point-sphere for each vertex using the CREATEVERTEX operator. Let $\mathcal{V}$ be the array of point-spheres thus created.

2. Read all faces. Parse each face and create a list of edges present in each face using the index into the array $\mathcal{V}$ to denote the edge ends.

3. Go through the list of edges created in step 2 and insert each edge. If an edge already exists between two vertices it need not be inserted again. If an edge would cause a self-loop it is not inserted.

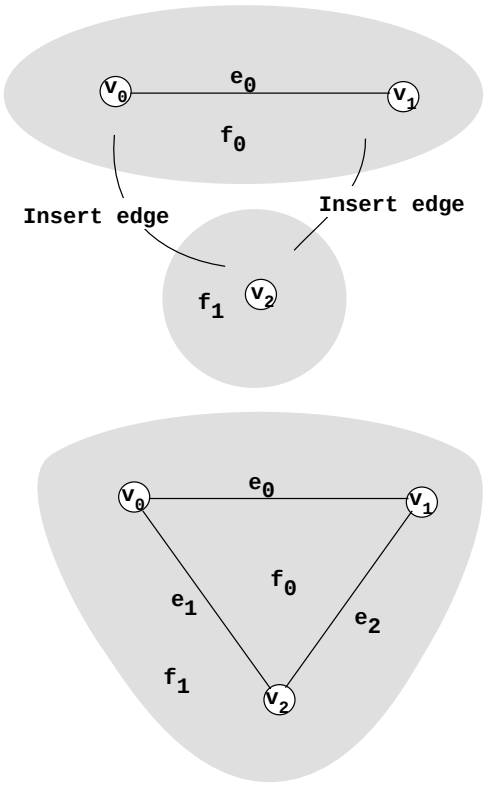

Figure 5. A triangle manifold (two-sided triangle) can be obtained by combining three point spheres with three insert edge operations. A face manifold (two sided face) can be obtained by combining n number of point spheres with n number of inserted edges.

The edges created in step 2 will be represented as a pair of corners. Note that corners in different faces can share the same vertex. Each vertex stores a list of corners that refer to it.

The edge insertion process uses the INSERTEDGE operator which inserts an edge between two corners. To insert an edge we first need to find the two end corners. Initially we have only a group of distinct point-spheres each corresponding to a vertex in the model and each containing one corner. Before any edges are inserted all these corners are tagged with a marker.

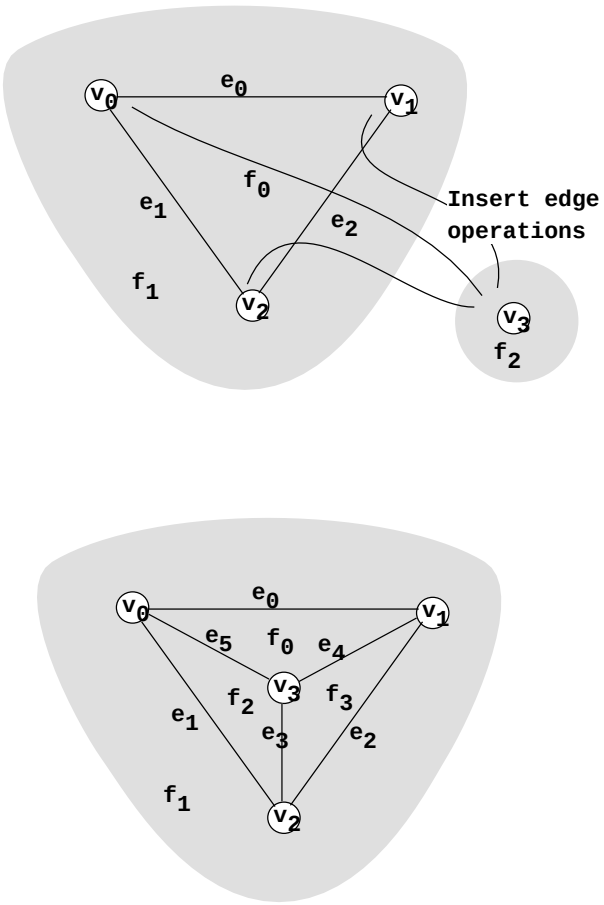

Figure 6. A tetrahedron can be obtained by combining four point spheres with six insert edge operations.

Let the two ends of edge $e_{1,2}$ be the corners $c_1 = \{a, v_1, b\}$ and $c_2 = \{c, v_2, d\}$ referring to vertices v_1 and v_2 respectively. (As we have mentioned earlier, a corner can be considered as a sequence of three vertices belonging to a face except when the face is a point sphere.) Inserting an edge involves the following steps:

1. Get the list of corners referring to vertex v_1 and v_2. We will refer to these lists as C_{v_1} and C_{v_2} corresponding to v_1 and v_2 respectively. To find the corner corresponding to c_1, go through C_{v_1} and find the corner which is preceded by vertex a or followed by vertex b. If C_{v_1} contains only 1 corner select that corner. The corner corresponding to c_2 is found similarly using C_{v_2}. If either of C_{v_1} or C_{v_2} contains more than one corner and a matching corner for c_1 or c_2 cannot be found, postpone insertion of this edge.

2. If we can find matching corners for both c_1 and c_2 in the above step, do the actual edge insertion using the INSERTEDGE operator.

After the first pass, we go through the list of postponed edges and repeat the above steps again.

If in two successive passes no postponed edges could be resolved, it indicates that the object has multiple 2-manifold interpretations. We choose one of the interpretations arbitrarily[1] by picking the originally tagged corner during the next pass. Note that this might need to be done for only one of the postponed edges since other edges might be resolved after this edge is inserted.

Once no more edges are left to be inserted the algorithm terminates.

4 Implementation and Testing

Our algorithm is implemented in C++ and has been incorporated into an existing manifold mesh modeling system. One of the main uses of the algorithm is to ensure thatWavefront obj files are always imported as correct manifolds. Corrupt files such as the ones representing non-manifolds or manifolds with boundaries could crash a manifold modeler. The new reader guarantees that we always deal with manifold meshes.

Our algorithm has been successfully tested on several publicly available models ranging in size from a few polygons to several thousands. We found that in practical examples the algorithm appears to be $O(nlogn)$ in terms of the number of edges.

[1]Note that this arbitrary choice depends only on the order of faces in the face list. In other words, there is a unique manifold interpretation for any given face list order. Thus, a given input will always give the same output.

4.1 Manifolds

Manifold mesh construction from manifold data is straightforward. It can easily be seen that if the arbitrary collection of polygons corresponds to a valid manifold, then the algorithm will create the corresponding manifold mesh. Because each edge of the manifold will be inserted exactly once, and each edge insertion will be consistent with the "correct" manifold representation, the resulting manifold will be the one described by the input data.

4.2 Crack Repair for Manifolds with Boundaries

How the algorithm repairs cracks can be understood by looking at Figure 1. As shown in the figure, all the cracks (i.e. missing polygons) are enclosed by boundary edges. In our algorithm, when we insert these boundary edges, the missing faces bounded by those edges are automatically created.

When enough edges are inserted to create a polygon, two faces are effectively created, a "front" face and a "back" face (see Figure 5). This back face can be thought of as filling in the empty area (i.e. the crack) around that polygon. As additional edges are inserted, this back face continues to fill in the regions not yet inserted into the model. Note that there may be several of these back faces existing simultaneously. Once all edges have been inserted, these back faces thus repair the cracks that might exist in the input data. Note that the orientation of these back faces will always be consistent with the rest of the model, as one would like.

We give below a simply example to demonstrate how crack repair works. Let the input data be a tetrahedron with a missing face and the faces be given as $f_0 = \{v_0, v_1, v_2\}$, $f_1 = \{v_0, v_3, v_1\}$ and $f_2 = \{v_0, v_2, v_3\}$. The algorithm first inserts edges for the face f_0, $\{v_0, v_1\}$ $\{v_1, v_2\}$ and $\{v_2, v_0\}$. After inserting these 3 edges a "back" face $f_b = \{v_0, v_2, v_1\}$ will be automatically created. Then, the algorithm inserts remaining edges for face f_1, which are $\{v_0, v_3\}$ and $\{v_3, v_1\}$. As a result, the back face now becomes $f_b = \{v_0, v_2, v_1, v_3\}$. In the next step, the algorithm adds remaining edges for the face f_2, which involves adding only the edge $\{v_2, v_3\}$. Inserting this edge divides f_b into two. One part becomes the face f_2, while the remaining back face is $f_b = \{v_2, v_1, v_3\}$. Notice that f_b now fills in, with correct orientation, the crack created by the missing face of the tetrahedron.

4.3 Non-Manifold Conversion and Insight to Manifolds

The interpration of non-manifold cases are more interesting. Using the reader, we can convert non-manifolds to manifolds. This conversion gives us a powerful insight into

the nature of manifolds. Some of the manifold shapes we create using this conversion sometimes defy conventional wisdom.

We present three examples illustrated in Figures 7, 8 and 9. As will be clear in all three examples, in order to convert a non-manifold to a manifold, our algorithm combines some of the faces to create larger faces. The conversion maintains the integrity of the edge directions in the combined faces.

In each of these figures, the leftmost image shows the non-manifold shape. The middle column shows some manifold interpretations of the non-manifold shape. The rightmost column shows the result of two iterations of Doo-Sabin subdivision applied to the manifold representations in the middle column. The Doo-Sabin smoothed versions are included (1) to demonstrate that non-manifold geometry has been converted to manifold, since Doo-Sabin can only be applied to manifold meshes and (2) to visually show that genus can vary, i.e. different manifold interpretations can have different genera.

The first example is shown in Figure 7. This nonmanifold shape is given by the following faces that are represented by cyclic ordered sets:

$f_0 = \{3,2,1,4\}$ $f_1 = \{7,8,9,10\}$
$f_2 = \{2,3,6\}$ $f_3 = \{4,1,5\}$
$f_4 = \{6,9,8\}$ $f_5 = \{5,7,10\}$
$f_6 = \{6,5,10,9\}$ $f_7 = \{5,6,3,4\}$
$f_8 = \{6,8,7,5\}$ $f_9 = \{1,2,6,5\}$

If the faces are given with the order above, our algorithm converts this non-manifold to a manifold by combining faces f_8 and f_9 into one hexagonal face $f_{10} = \{1,2,6,8,7,5\}$ as shown in top row of Figure 7. Note that the hexagon f_{10} is automatically created once we insert all the edges. More interesting cases can occur when the algorithm reads the faces in different orders. One such case is shown in the bottom row of Figure 7. In this case the faces f_4, f_5, f_7 and f_9 are combined into one face $f_{11} = \{9,8,6,3,4,5,7,10,5,1,2,6\}$. This large face is self intersecting and makes the genus of the object 1 as can clearly be seen in the Doo-Sabin smoothed image.

The second example is shown in Figure 8. This nonmanifold shape is given by the following faces that are represented by cyclic ordered sets:

$f_0 = \{3,2,1,4\}$ $f_1 = \{6,7,8,9\}$
$f_2 = \{1,2,5\}$ $f_3 = \{7,6,5\}$
$f_4 = \{4,1,5\}$ $f_5 = \{6,9,5\}$
$f_6 = \{3,4,5\}$ $f_7 = \{9,8,5\}$
$f_8 = \{2,3,5\}$ $f_9 = \{8,7,5\}$

If the faces are given with the order above, our algorithm converts this non-manifold to a manifold by combining faces f_8 and f_9 into one hexagonal face $f_{10} = \{2,3,5,8,7,5\}$ as shown in top row of Figure 8. The second and third rows show two other interpretations when the algorithm reads the faces in different orders. In the middle row, f_9 and f_2 are read as the last two polygons and the algorithm combines them into a new face $f_{11} = \{1,2,5,8,7,5\}$. In the last row, f_9 and f_4 are read as the last two polygons and the algorithm combines them into a new face $f_{12} = \{4,1,5,8,7,5\}$. In this example it is clear that the algorithm treats the last two polygons as if they are missing.

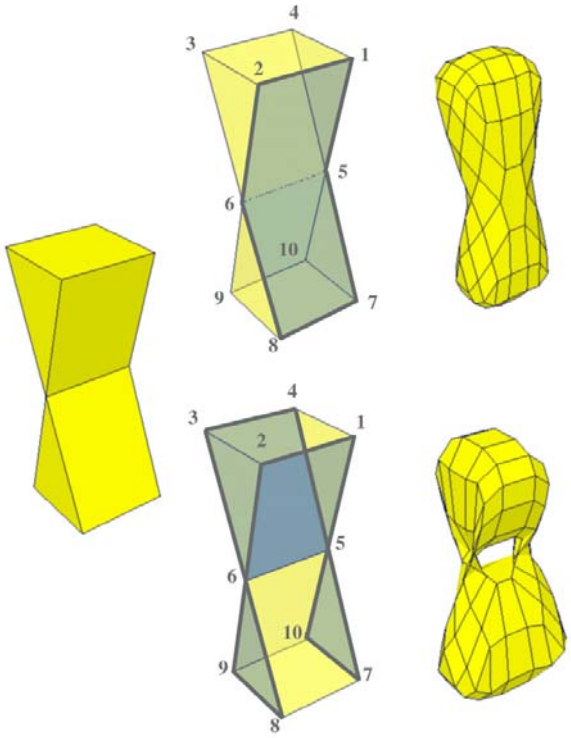

Figure 7. Two 2-manifold interpretations of a non-manifold.

In the next example, we have introduced an aditional face inside of a prism as shown in Figure 9. This shape is given by the following faces that are represented by cyclic ordered sets.

$f_0 = \{4,3,2,1\}$ $f_1 = \{9,10,11,12\}$
$f_2 = \{7,8,12,11\}$ $f_3 = \{3,4,8,7\}$
$f_4 = \{4,1,5,8\}$ $f_5 = \{8,5,9,12\}$
$f_6 = \{2,3,7,6\}$ $f_7 = \{6,7,11,10\}$
$f_8 = \{1,2,6,5\}$ $f_9 = \{5,6,10,9\}$
$f_{10} = \{5,6,7,8\}$

Since this visual example is more complicated than earlier ones, we have not drawn the face f_{10} and the initial shape looks like a simple prism. If the faces are given with the order above, our algorithm simply eliminates the face f_{10} as shown in the top row of Figure 9. The more interesting example is shown in the bottom row of Figure 9. In this case, several faces are combined. Since more than one

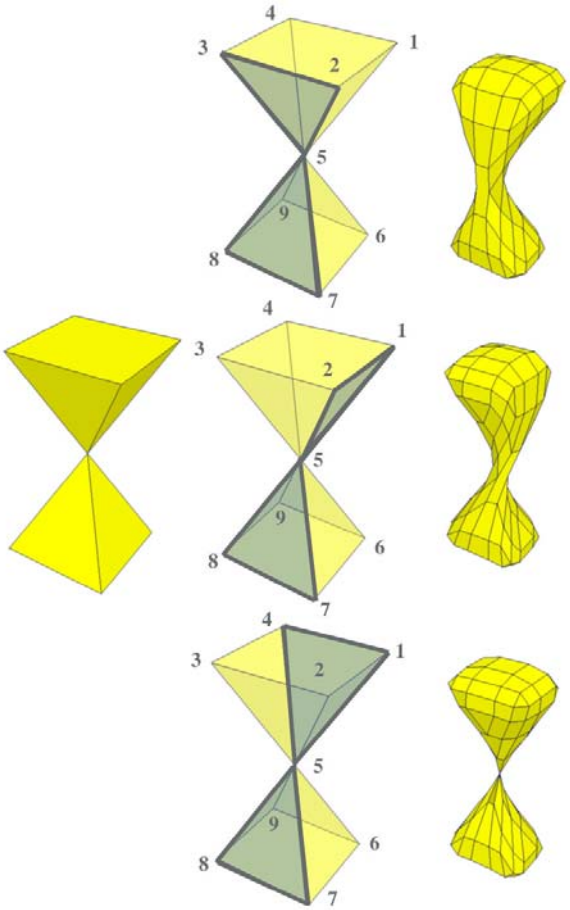

Figure 8. Three 2-manifold interpretations of a non-manifold.

combined face is created we have not highlighted them to avoid confusion. The following information can be helpful to understand the manifold structure shown in the bottom row:

- f_4 and f_5 combine and become a hexagon as $\{4, 1, 5, 9, 12, 8\}$.
- f_6 and f_7 combine and become a hexagon as $\{2, 3, 7, 11, 10, 6\}$.
- f_3, f_8 and f_{10} combine and become one octagon as $\{3, 4, 8, 5, 1, 2, 6, 7\}$.

Although we do not give any examples here, we tested our algorithm on models that include some wrongly-oriented faces. As in the case of non-manifold input data, the conversion maintains the integrity of the edge directions by combining two or more faces into one.

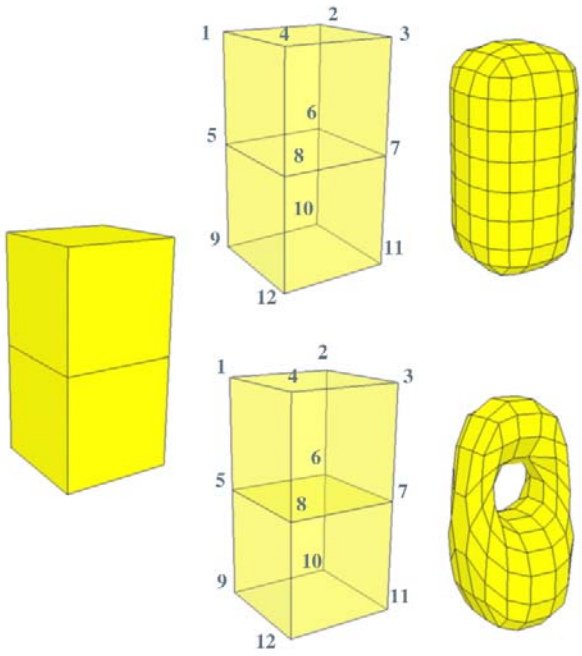

Figure 9. Two 2-manifold interpretations for a non-manifold.

5 Conclusion and Future Work

Manifolds with boundaries, non-manifolds, and artifacts are very common in computer graphics files. Many modeling packages allow manifolds with boundaries and non-manifolds. Also, many practical mesh data structures allow arbitrary collections of polygons that can include artifacts such as wrongly-oriented polygons and T-junctions.

Various algorithms have been developed for the identification and correction of such artifacts [6, 16]. These methods are often quite complicated and rely on geometric information. The algorithm we have presented is different from others in that it is simple, is purely topological, and theoretically guarantees the manifold property of the resulting mesh regardless of the input data.

Note that our algorithm totally ignores the geometric properties of the underlying shape. For instance, self intersections are not considered and cannot be eliminated by our algorithm. Moreover, two vertices are considered different if they are declared separately regardless of their position (i.e. even if their positions are the same, they are considered different). Ignoring the geometry greatly simplifies our algorithm.

5.1 Future Work

Our algorithm is useful in other aspects of modeling system implementation, beyond file input or transfer. For example, our technique has been used to create the dual of any 2-manifold mesh. Ignoring twogons and self-loops is extremely useful for file input or transfer since we do not create unnecessary edges. However, if this algorithm is used for other operations, this restriction can be a problem. One problem with the dual operation is that if the initial mesh has valence-2 or valence-1 vertices, the dual cannot correctly be created since it should have twogons and self-loops. Thus, in order to use the algorithm to simplify programming tasks, we would need to include twogons and self-loops.

Including geometry can be another improvement. For instance, all the shapes in Figure 8 are the same in terms of topology. However, each shape gives a different subdivided shape. By augmenting our strictly topological methods with some geometric constraints, we could provide users additional control.

References

[1] E. Akleman and J. Chen, Guaranteeing the 2-manifold property for meshes with doubly linked face list, *International Journal of Shape Modeling*, Volume 5, no. 2, pp. 149-177, 1999.

[2] E. Akleman, J. Chen, and V. Srinivasan, A new paradigm for changing topology during subdivision modeling, *Proceedings 8th Pacific Conference on Computer Graphics and Applications*, (PG'2000), (2000), pp. 192-201.

[3] E. Akleman, J. Chen, V. Srinivasan and F. Eryoldas, A New Corner Cutting Scheme with Tension and Handle-Face Reconstruction, *International Journal of Shape Modeling*, No. 2, pp. 111-121, 2001.

[4] E. Akleman, J. Chen and V. Srinivasan, An interactive system for modeling 2-manifold meshes, *Proceedings of International Conference on Shape Modeling and Applications 2002*, pp. 43-50, May 2002.

[5] J. Chen and E. Akleman, Topologically robust mesh modeling: concepts, structures and operations, Submitted to a journal (See http://www-viz.tamu.edu/faculty/ergun/research/topology/ to get a copy).

[6] G. Barequet and S. Kumar, "Repairing CAD models", in *Proceedings of IEEE Visualization'97*, (October 1997) pp. 363-370.

[7] B. J. Baumgart, "Winged-edge polyhedron representation", Technical Report CS-320, Stanford University, 1972.

[8] J. Chen, "Algorithmic Graph Embeddings", *Theoretical Computer Science*, **181** (1997) 247-266.

[9] D. Doo and M. Sabin, "Behavior of Recursive Subdivision Surfaces Near Extraordinary Points", *Computer Aided Design*, **10** (September 1978) 356-360.

[10] J. Edmonds, "A Combinatorial Representation for Polyhedral Surfaces", *Notices American Mathematics Society*, **7** (1960) 646.

[11] J. L. Gross and T. W. Tucker, *Topological Graph Theory*, (Wiley Interscience, New York, 1987).

[12] L. Guibas, J. Stolfi, "Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams", *ACM Transaction on Graphics*, **4** (1985) 74-123.

[13] L. Heffter, "Uber das Problem der Nachbargebiete", *Math. Annalen*, **38** (1891) 477-508.

[14] C. M. Hoffmann, *Geometric & Solid Modeling, An Introduction*, (Morgan Kaufman Publishers, Inc., San Mateo, Ca., 1989).

[15] M. MÄNTYLÄ, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MA, 1988.

[16] T. M. Murali and T. A. Funkhouser, "Consistent solid and boundary representations from arbitrary polygonal data", in *Proceedings of 1997 Symposium on Interactive 3D Graphics* , (1997) pp. 155-162.

[17] D. Zorin and P. Schröder, co-editors, *Subdivision for Modeling and Animation*, ACM SIGGRAPH'2000 Course Notes no. 23, July, 2000.