International Journal of Shape Modeling, Vol. 5, No. 2 (1999) 159–177

GUARANTEEING THE 2-MANIFOLD PROPERTY FOR MESHES WITH DOUBLY LINKED FACE LIST

ERGUN AKLEMAN
Department of Architecture, Texas A&M University
College Station, Texas 77843-3137, United States of America

and

JIANER CHEN
Department of Computer Science, Texas A&M University
College Station, Texas 77843-3112, United States of America

Received (May 10, 1999)
Revised (July 26, 1999)
Communicated by (Alexander Pasko)

Meshes, which generalize polyhedra by using non-planar faces, are the most commonly used objects in computer graphics. Modeling 2-dimensional manifold meshes with a simple user interface is an important problem in computer graphics and computer aided geometric design. In this paper, we propose a conceptual framework to model meshes. Our framework guarantees topologically correct 2-dimensional manifolds and provides a new user interface paradigm for mesh modeling systems.

Keywords: Computer Aided Geometric Design, Shape Modeling, Computer Graphics, Visualization, Topology.

1. Introduction and Motivations

Most current shape modeling systems are based on tensor product parametric surfaces such as Tensor Product B-splines [1]. However tensor product surfaces can only support quadrilateral meshes as control meshes for smooth surfaces, and do not provide general topology [2, 3]. The restriction to quadrilateral meshes also makes the modeling process difficult for users. There have been significant efforts in solving this problem by still staying in the parametric surface realm [2, 3]. On the other hand, subdivision surfaces, which were originally proposed as a generalization of B-spline surfaces [4, 5], have recently resurfaced as an alternative for providing general topologies [6, 7].

Subdivision surfaces assume that the users provide an irregular control mesh. These initial control meshes can either be created by direct modeling or obtained by scanning a sculpted real object. A smoother version of this initial mesh without changing the original topology is obtained by subdivision operations. The effective

usage of subdivision surfaces requires that the initial control meshes represent a valid and correct 2-dimensional manifold in 3-D space. Since the quality and topology of the smooth surface resulting from subdivision rules greatly depend on the initial control mesh, theoretical assurance of the quality of initial control meshes is extremely important. In other words, the process of obtaining the initial control mesh must be robust and guarantee valid 2-dimensional manifolds. Unfortunately, set operations, which are the most commonly used operations in mesh modeling, can result in non-manifold surfaces. Moreover, the existing data structures in mesh modeling are specifically developed in such a way that they can represent non-manifold surfaces resulting from the set operations. In particular, they do not guarantee valid 2-dimensional manifold surfaces. Because of this fundamental problem, in the process of obtaining the initial control mesh, unwanted artifacts can be generated. These artifacts include wrongly-oriented polygons, intersecting or overlapping polygons, missing polygons, cracks, and T-junctions. There have been recent research efforts to correct these artifacts [8, 9].

Besides guaranteeing topological consistency, data structures for mesh modeling should also support topological operations efficiently.

The classical view of mesh representation is based on adjacency relationships between points, edges and faces. For instance, the vertex-edge adjacency relationship specifies two adjacent vertices for each edge. There exist nine such adjacency relationships, but it is sufficient to maintain only three of the ordered adjacency relationships to obtain the others [10].

In most practical computer graphics applications, meshes are often represented with one adjacency relationship. The data structure is generally organized as an unordered list of polygons where each polygon is specified by an ordered sequence of vertices, and each vertex is specified by its x, y, and z coordinates [8]. Let us call this data structure a *vertex-polygon list.* Vertex-polygon lists do not always guarantee topological consistency. In addition, they can even create degeneracies such as cracks, holes and overlaps [8, 9].

These degeneracies can be partly eliminated by adding an additional adjacency relationship: edge lists to vertex-polygon lists [8]. In a *vertex-polygon-edge list* structure, a list of vertices, a list of directed edges, and a list of polygons are described. Vertices are specified by their three coordinates, directed edges are specified by two vertices, and polygons are specified by an ordered sequence of edges. Each polygon is oriented in a consistent direction, typically counter-clockwise when viewed from outside of the model. Because of the last condition, vertex-polygon-edge lists are more powerful than vertex-polygon lists. However, the representation does not guarantee valid manifold surfaces either. It is still possible to specify a non-manifold surface by giving wrong specifications in terms of the vertex-polygon-edge list.

One of the oldest formalized data structures that supports manifold surfaces is the *winged-edge* representation [11]. Winged-edge data structures support 2-dimensional manifold surfaces, but like vertex-polygon-edge lists they can also accept non-manifold surfaces [11, 12].

Baumgart suggested using a winged-edge structure and Euler operators in order to obtain coordinate free operations [13]. Guibas and Stolfi introduced the *quad-edge* data structure and topological operators such as splice operators [14].

When using set operations, resulting solids can have non-manifold boundaries [15, 16, 17, 18]. It is worthwhile to note that although the data structures, such as winged-edge, can handle some non-manifold surfaces, they actually complicate the algorithms for solid modeling [16, 19]. Data structures that can support a wider range of non-manifold surfaces have been investigated. Examples of such work are Weiler's *radial-edge* structure [20], Karasick's *star-edge* structure [21], and Vanecek's *edge-based* data structure [22].

In the current paper, we propose to return back to the basic concept of coordinate free operations over 2-dimensional manifold surfaces by ignoring set-operations. Similar to our approach, instead of set operations the usage of Morse operators that describe the changes of cross-sectional contours at critical sections (peaks, passes and pits) has recently been investigated [24, 23]. We use topological graph operations which are similar to Euler's operations and based on graph embeddings. The biggest advantage of our operations is that they are extremely simple and always guarantee topological consistency. Only two operations, INSERT(edge) and DELETE(edge), are enough to change the topology. If an inserted or deleted edge adds or removes a handle, we can efficiently find the new topology by using graph embeddings. This efficient computation is due to our *Doubly Linked Face List (DLFL)* data structure [31]. We propose to use this data structure to support a representation in which the basic topological operations related to computer graphics, such as surface subdivision, adding or removing a handle, can all be done very efficiently.

DLFL not only supports efficient computations on 2-dimensional manifolds, but also always guarantees topological consistency, i.e. it always gives a valid 2-dimensional manifold. In addition, DLFL uses the minimum amount of computer memory.

In order to provide an intuitive interface to users for manipulating the topology of 2-dimensional manifolds, we add parameter coordinates into vertex specifications. These parameter coordinates come from the polygonal representations of orientable 2-dimensional manifolds which are $4n$-sided polygons that are obtained from transforming the torus with n handles and $2n$ cuts [25]. We also show that the polygonal representation of 2-dimensional manifolds is efficiently supported by our DLFL structure.

The combination of the graph rotation system representation and the polygonal representation of meshes provides a new approach for defining modeling meshes.

- The topology of meshes can be updated automatically during the modeling process simply by adding or removing new vertices and edges to the graph.
- A new topology can be obtained by merging two polygonal representations of two 2-dimensional manifolds.

- The polygonal representation can be used for texture coordinates. By changing the parametric coordinates of the vertices, users can change the texture mapping. Moreover, the users can directly paint in the parameter space in order to create a texture.

- Our new proposed data structure will be extremely useful in the computation of the subdivision surface when the mesh is used as a subdivision surface control mesh.

2. New Representations and Their Data Structures

In this section, we introduce the fundamentals of graph embeddings and graph rotation systems, and present a new data structure for them, which provides a new representation for 2-dimensional manifold mesh modeling.

2.1. *Graph Rotation System*

The graph rotation system is a powerful tool for guaranteeing topological consistency. In this subsection, we introduce historical background and some mathematical fundamentals (see [26] for more detailed discussion).

The concept of rotation systems of a graph originated from the study of graph embeddings and it is implicitly due to Heffter [27] who used it in Poincare dual form. A graph embedding in an orientable surface corresponds to an obvious rotation system, namely, the one in which the rotation at each vertex is consistent with the cyclic order of the neighboring vertices in the embedding. Edmonds [28] was the first to call attention explicitly to studying rotation systems of a graph.

Let G be a graph. A *rotation* at a vertex v of G is a cyclic permutation of the edge-ends incident on v. A *rotation system* of G is a list of rotations, one for each vertex of G. Given a rotation system of a graph G, to each oriented edge (u, v) in G, one assigns the oriented edge (v, w) such that vertex w is the immediate successor of vertex u in the rotation at vertex v. The result is a permutation on the set of oriented edges, that is, on the set in which each undirected edge appears twice, once with each possible direction. In each edge-orbit under this permutation, the consecutive oriented edges line up head to tail, from which it follows that they form a directed cycle in the graph. If there are r oriented edges in an orbit, then an r-sided polygon can be fitted into it. Fitting a polygon to every such edge-orbit results in polygons on both sides of each edge, and collectively the polygons form a 2-dimensional manifold.

For example, consider the graph G given in Figure 1, where G is drawn in such a way that the rotation at each vertex can be traced by traversing the incident edges in clockwise order. More specifically, the rotation system of G is:

$$\begin{array}{lll} a: a'cdb & b: b'aec & c: c'bfa \\ d: d'eaf & e: e'fbd & f: f'dce \end{array}$$

$$a' : ab'd'c' \qquad b' : bc'e'a' \qquad c' : ca'f'b'$$
$$d' : df'a'e' \qquad e' : ed'b'f' \qquad f' : fe'c'd'$$

This rotation system has twelve orbits, which are given as follows:

$$O_1 = adfc \qquad O_2 = dabe \qquad O_3 = ebcf$$
$$O_4 = a'd'e'b' \qquad O_5 = d'a'c'f' \qquad O_6 = f'c'b'e'$$
$$O_7 = dd'f'f \qquad O_8 = d'dee' \qquad O_9 = ff'e'e$$
$$O_{10} = aa'b'b \qquad O_{11} = a'acc' \qquad O_{12} = bb'c'c$$

If we associate each orbit with a 4-sided polygon, we obtain an embedding of the graph G on the torus (i.e., the 2-dimensional manifold of genus 1), as shown in Figure 2.

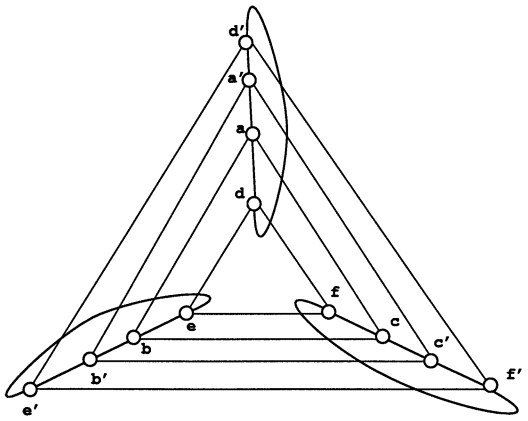

Fig. 1. A graph G drawn in a rotation system

Edmonds [28] has shown that every rotation system of a graph gives a *unique* orientable 2-dimensional manifold. Moreover, the corresponding orientable 2-dimensional manifold is constructible, as we described above. For any 2-dimensional manifold S, there is a graph G and a rotation system ρ of G such that ρ corresponds to an embedding of G on S [29].

Therefore, the existence of the bijective correspondence between graph embeddings on orientable 2-dimensional manifolds and graph rotation systems enables us to represent topological objects by combinatorial ones. In particular, every 2-dimensional manifold can be represented by a rotation system of a graph, and every rotation system of a graph corresponds to a valid 2-dimensional manifold. In consequence, the presentation of graph rotation systems always guarantees topological consistency. Recently, we have developed a very efficient algorithm that, given a

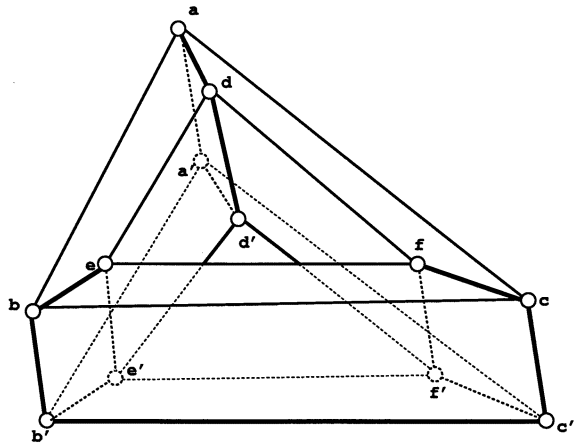

Fig. 2. The embedding of the graph G corresponding to the rotation system in Figure 1

rotation system of a graph, constructs the corresponding 2-dimensional manifold [30].

We should also point out that each edge of a graph appears exactly twice in any of its rotation systems. Therefore, the amount of computer memory used to store a graph rotation system is very small.

2.2. *The Doubly Linked Face List*

Graph rotation systems are an effective tool for representing 2-dimensional manifolds. We have identified the following topological operations on rotation systems as essential for 2-dimensional manifold mesh modeling:

- FACE-TRACE(f) outputs a boundary walk of the face f. This is related to reconstructing a polygon in a 2-dimensional manifold and to triangulation of a polygon in a 2-dimensional manifold.

- VERTEX-TRACE(v) outputs the edges incident on the vertex v in the (circular) ordering of the rotation at v. This is useful when a 2-dimensional manifold needs reshaping.

- INSERT(c_1, c_2, e) inserts the new edge e between the face corners c_1 and c_2 (a face corner is a subsequence of a face boundary walk consisting of two consecutive edges plus the vertex between them). This is used in triangulation of a polygon in a 2-dimensional manifold, changing the topology of a 2-dimensional manifold (i.e., adding a handle to a manifold), and gluing two separated manifolds. Figure 3 shows two examples of edge insertion. The first insertion does not change the topology, but the second insertion changes the topology.

- DELETE(e) deletes the edge e from the current embedding. This is the converse operation of edge insertion and is also used in changing the topology of a manifold.
- COFACIAL(c_1, c_2) returns *true* if the two face corners c_1 and c_2 belong to the same face of the current embedding and *false* otherwise. This operation is useful in maintaining topological consistency of manifolds.
- SUBDIV(e, v) subdivides the edge $e = (u, w)$ by a new vertex v of degree 2 so that the edge e becomes two new edges (u, v) and (v, w). This is the operation that increases the number of vertices in a mesh.

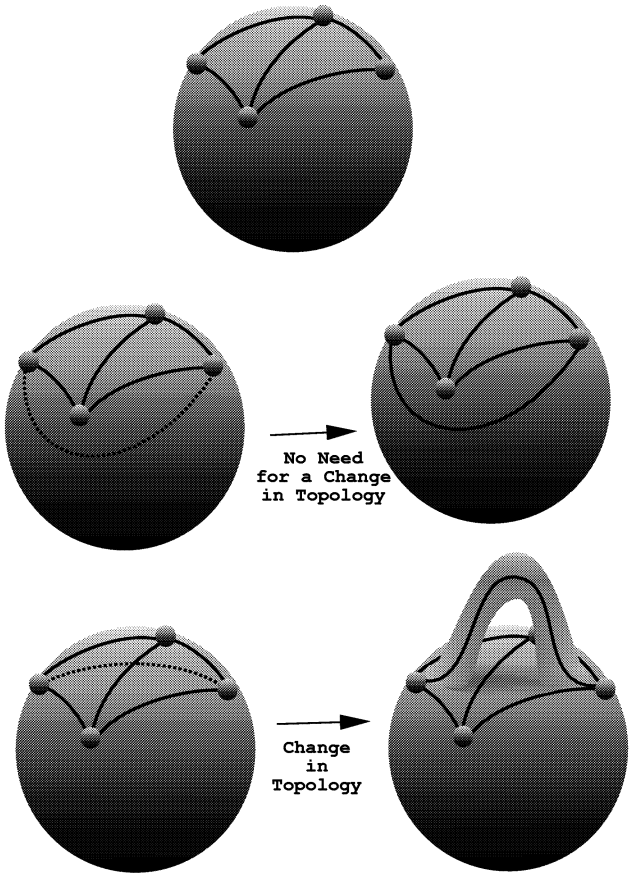

Fig. 3. Topological effect of edge insertion in a rotation system.

We have initiated the study of algorithms on graph rotation systems in order to implement these operations efficiently [31]. We have first observed that if a rotation system of a graph is represented in the *edge-list form*, which for each vertex v

contains the list of its incident edges, arranged in the order according to the rotation at v, the representation does not efficiently support the operations listed above.

Another data structure, which is closely related to the winged-edge structure, *the doubly-connected-edge-list* (DCEL) has been widely used in computational geometry [32]. DCEL can also be directly used for representing a rotation system. However, we observed that the DCEL structure does not support the operations INSERT and DELETE efficiently [31].

We introduce a new data structure and show that the new data structure efficiently supports all of the operations listed above.

Each face is given by a sequence of vertices corresponding to a boundary traversing of the face. The vertex appearances in the sequence will be called *vertex nodes.* Note that two consecutive vertex nodes in the sequence correspond to an edge side in the embedding. The sequence is represented by a cyclically concatenatable data structure. For specific discussion, we will use 2-3 trees for this concatenatable data structure [33]. For readers' convenience, we recall that a 2-3 tree is a balanced tree whose depth is always logarithmic in the number of nodes in the tree. Moreover, operations on 2-3 trees such as inserting a node, deleting a node, splitting a 2-3 tree into two 2-3 trees, and concatenating two 2-3 trees into a single 2-3 tree, can all be done in logarithmic time (see [33] for more detailed discussion).

Definition 1 *Let $\rho(G)$ be an embedding of a graph $G = (V, E)$ with face set F. A doubly-linked-face-list (DLFL) for the embedding $\rho(G)$ is a triple $L = \langle \mathcal{F}, \mathcal{V}, \mathcal{E} \rangle$, where the face list $\mathcal{F}$ consists of a set of $|F|$ sequences. Each is given by a 2-3 tree and corresponds to the boundary walk of a face in the embedding $\rho(G)$. Moreover, the roots of the 2-3 trees are connected by a circular doubly linked list. The vertex array $\mathcal{V}$ has $|V|$ items. Each $\mathcal{V}[v]$ is a linked list of pointers to the vertex nodes of v in the 2-3 trees in $\mathcal{F}$. The edge array $\mathcal{E}$ has $|E|$ items. Each $\mathcal{E}[e]$ is doubly linked to the first vertex nodes of the two edge sides of the edge e in the 2-3 trees in $\mathcal{F}$.*

Figure 4 gives an illustration of the DLFL data structure for a tetrahedron. It can be shown that the DLFL structure and the DCEL structure used in computational geometry can be converted from one to the other in linear time [31]. This implies that the computer space used by a DLFL structure to represent a mesh modeling is linear in the size of the mesh, which is the best possible.

Now we discuss how the above listed operations for mesh modeling are implemented on the DLFL structure.

The operations FACE-TRACE(f) and VERTEX-TRACE(v) are given in Figure 5. Recall that in the DLFL structure, each face f is specified by the face array element $\mathcal{F}[f]$ and each vertex v is specified by the vertex array element $\mathcal{V}[v]$.

Theorem 1 *Based on the DLFL structure, the operation* FACE-TRACE(f) *can be done in time linear in the size of the face f, and the operation* VERTEX-TRACE(v) *can be done in time linear in the degree of the vertex v.*

Proof. The FACE-TRACE algorithm is basically just the traversing algorithm for the 2-3 tree representing the face f. It is well known that its time complexity is

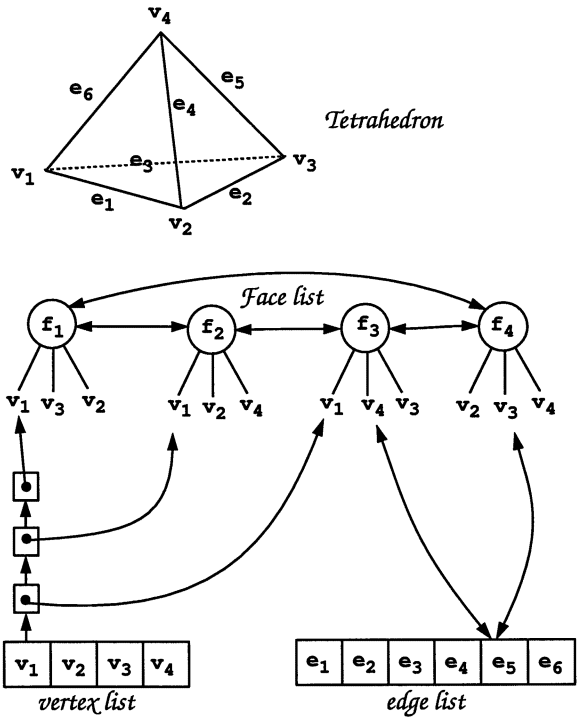

Fig. 4. An illustration of the DLFL data structure for a tetrahedron.

FACE-TRACE($\mathcal{F}[f]$).

1. r = the root of the 2-3 tree for the face f;
2. Traverse(r).

Traverse(r).

1. **if** r is a leaf **then** print(r), **return**;
2. **for** each child u of r **do** Traverse(u).

VERTEX-TRACE($\mathcal{V}[v]$).

1. pick a pointer in the list $\mathcal{V}[v]$ to find an edge side (v, w_0); print(v, w_0);
2. find the 2-3 tree T in $\mathcal{F}$ that contains the edge side (w_0, v);
3. let (v, w) be the edge side following (w_0, v) in the 2-3 tree T;
4. **while** $(v, w) \neq (v, w_0)$ **do**
   print(v, w);
   find the 2-3 tree T in $\mathcal{F}$ that contains the edge side (w, v);
   let (v, w') be the edge node following (w, v) in the 2-3 tree T;
   $w = w'$.

Fig. 5. FACE-TRACE and VERTEX-TRACE

CoFacial(c_1, c_2)

1. find the root r_1 of the 2-3 tree containing the face corner c_1;
2. find the root r_2 of the 2-3 tree containing the face corner c_2;
3. **return** $(r_1 = r_2)$.

Fig. 6. CoFacial algorithm

linear in the size of the tree, thus also linear in the size of the corresponding face f.

We need some explanations on the algorithm Vertex-Trace(v). By the structure of the DLFL, each vertex array element $\mathcal{V}[v]$ is a list of pointers to the vertex nodes labeled with v in the 2-3 trees. Therefore, picking any pointer in $\mathcal{V}[v]$ gives a vertex node for v, and the vertex node w_0 immediately following this vertex node makes an edge side (v, w_0). Since the edge $\mathcal{E}[e]$ with two ends v and w_0 is doubly linked to the first vertex nodes of the two edge sides (v, w_0) and (w_0, v), in constant time we can find the vertex node of w_0 followed immediately by a vertex node of v. Therefore, the 2-3 tree T containing the edge side (w_0, v) can be found in constant time. Finally, note that if the edge side (v, w) follows right after the edge side (w_0, v) in the 2-3 tree T, then vertex w is right after vertex w_0 in the rotation at vertex v. Therefore, the Vertex-Trace algorithm correctly prints the edge (v, w) right after edge (v, w_0). For the same reason, the execution of the **while** loop in step 4 prints each of the edges incident on vertex v in the order of the rotation at the vertex v. Since we can always find in constant time the next edge in the rotation at the vertex v, we conclude that the Vertex-Trace takes time linear in the degree of the vertex v. □.

The CoFacial operation is simple. Note that a face corner can be given simply by a vertex node in the 2-3 trees. Given two face corners c_1 and c_2, we only need to check if they are in the same 2-3 tree in the face list $\mathcal{F}$. The algorithm is given in Figure 6.

To study the complexity of the operations Insert and Delete, let us have a closer look at these two operations.

First consider the operation Insert(c_1, c_2, e), where $e = (v_1, v_2)$ is an edge to be inserted between the face corner c_1 at vertex node v_1 and face corner c_2 at vertex node v_2. There are two possible cases.

If c_1 and c_2 are face corners of the same face f, then inserting the edge e will split the face f into two faces and unchange the topology. More precisely, suppose that the boundary walk of face f is $B_f = \alpha w_1 v_1 w_1' \beta w_2 v_2 w_2'$, where α and β are subwalks, then inserting the edge e will result in two faces with the boundary walks $B_f' = \alpha w_1 v_1 v_2 w_2'$ and $B_f'' = \beta w_2 v_2 v_1 w_1'$, respectively. This situation is illustrated in Figure 7. Therefore, the two new face boundary walks B_f' and B_f'' can be obtained by properly splitting the sequence B_f plus a couple of local modifications. Note that the 2-3 tree representing the face f may be a circular permutation of the sequence B_f. In this case, we also need to first split then re-concatenate the sequence properly

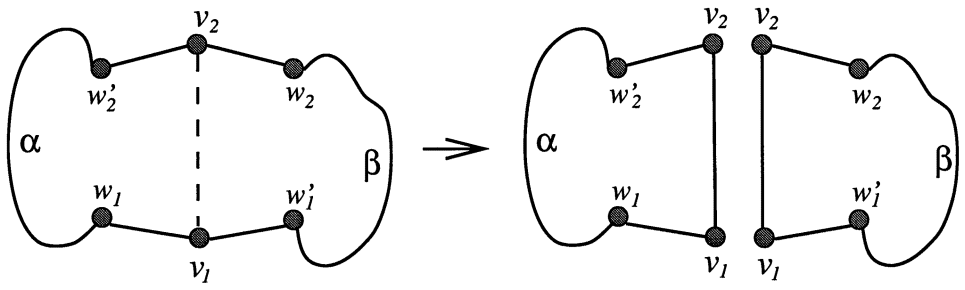

Fig. 7. Inserting an edge between two corners of the same face

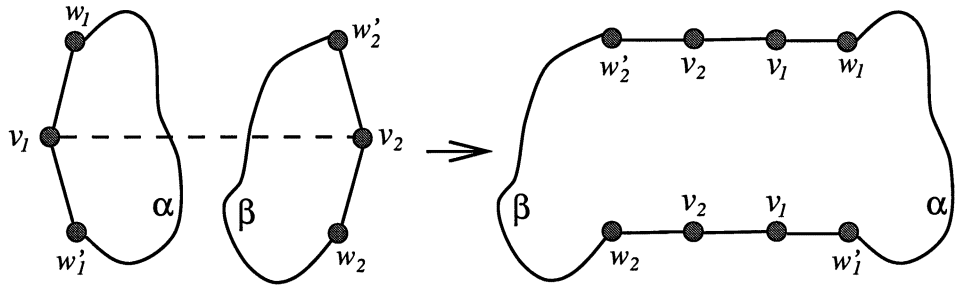

Fig. 8. Inserting an edge between two corners of the same face

to get the sequence B_f to start with.

If the face corners c_1 and c_2 belong to different faces $\alpha w_1 v_1 w_1'$ and $\beta w_2 v_2 w_2'$, where α and β are subwalks, then inserting the edge e will merge these two faces into a larger face

$$\alpha w_1 v_1 v_2 w_2' \beta w_2 v_2 v_1 w_1'$$

Figure 8 illustrates this situation. Note also that in this case, a handle is added to the manifold so that the genus of the manifold is increased by 1 (i.e., the topology of the manifold is changed). The sequence for this larger face can also be obtained by a number of proper splitting and concatenation operations.

Summarizing the above discussion, we give the algorithm for INSERT in Figure 9.

The DELETE operation can be done similarly. Given the edge $e = (v_1, v_2)$ to be deleted from the 2-dimensional manifold, we first get from the edge array element $\mathcal{E}[e]$ in the DLFL structure the two edge sides (v_1, v_2) and (v_2, v_1) of e and use COFACIAL operation to check whether the two edge sides belong to the same face in the embedding. If the two edge sides belong to different faces $\alpha v_1 v_2$ and $\beta v_2 v_1$ of the embedding, then deleting the edge e will merge the two faces into a single face $\alpha v_1 \beta v_2$ with the topology unchanged. If the two edge sides of e belong to the

INSERT(c_1, c_2, e)

1. assume the face corners c_1 and c_2 are at vertex nodes labeled by v_1 and v_2, respectively, where $e = (v_1, v_2)$;
2. **if** COFACIAL(c_1, c_2) **then**
   find the 2-3 tree B_f in $\mathcal{F}$ containing the face corners c_1 and c_2;
   { the next two statements make a circulation of the face boundary B_f so that the edge side (v_2, w_2') appears at the end of the sequence. }
   split the sequence B_f at w_2' into subsequences $B_1 = \delta v_2 w_2'$ and B_2;
   re-concatenate B_2 and B_1 into $B_f = B_2 \cdot B_1 = \alpha w_1 v_1 w_1' \beta w_2 v_2 w_2'$;
   split B_f at w_1' to two subsequences $\alpha w_1 v_1$ and $\beta w_2 v_2 w_2'$;
   split $\beta w_2 v_2 w_2'$ at w_2' to obtain $\beta w_2 v_2$;
   concatenate $\alpha w_1 v_1$, v_2, and w_2' to obtain the sequence $B_f' = \alpha w_1 v_1 v_2 w_2'$;
   concatenate $\beta w_2 v_2$, v_1, and w_1' to obtain the sequence $B_f'' = \beta w_2 v_2 v_1 w_1'$;
   else
   find the 2-3 tree $B_1 = \alpha w_1 v_1 w_1'$ in $\mathcal{F}$ containing the face corner c_1;
   find the 2-3 tree $B_2 = \beta w_2 v_2 w_2'$ in $\mathcal{F}$ containing the face corner c_2;
   split $\alpha w_1 v_1 w_1'$ at w_1' to obtain $\alpha w_1 v_1$;
   split $\beta w_2 v_2 w_2'$ at w_2' to obtain $\beta w_2 v_2$;
   concatenate $\alpha w_1 v_1$, v_2, w_2', $\beta w_2 v_2$, v_1, w_1' into $\alpha w_1 v_1 v_2 w_2' \beta w_2 v_2 v_1 w_1'$.

Fig. 9. INSERT algorithm

same face $\delta v_1 v_2 \gamma v_2 v_1$, then deleting the edge e will break the face into two faces δv_1 and γv_2 and decrease the embedding genus by 1 (thus change the topology). The DELETE algorithm is given in Figure 10.

Finally, the SUBDIV(e, v) operation is implemented in the DLFL structure by replacing the edge $e = (u, w)$ by two new edges (u, v) and (v, w), and introducing a new vertex v. The detailed algorithm is given in Figure 11.

Theorem 2 *The DLFL structure supports the operations* COFACIAL, INSERT, DELETE, *and* SUBDIV *in logarithmic time.*

Proof. It is well known that the depth of a 2-3 tree is logarithmic to the number

DELETE(e)

1. find the two edge sides (v_1, v_2) and (v_2, v_1) of e;
2. **if** the two edge sides of e belong to different faces $\alpha v_1 v_2$ and $\beta v_2 v_1$ **then**
   split $\alpha v_1 v_2$ at v_1 to obtain α;
   split $\beta v_2 v_1$ at v_2 to obtain β;
   concatenate α, v_1, β, and v_2 to obtain $\alpha v_1 \beta v_2$;
   else {the two sides of e belong to the same face $\delta v_1 v_2 \gamma v_2 v_1$}
   split $\delta v_1 v_2 \gamma v_2 v_1$ to obtain two new faces δv_1 and γv_2.

Fig. 10. DELETE algorithm

SUBDIV(e, v)

1. find the 2-3 trees B_1 and B_2 that contain the two sides (u, w) and (w, u) of the edge e;
2. insert v in B_1 between u and w and insert v in B_2 between w and u;
3. create a new vertex v in the vertex list, pointing to the two occurrences of v in B_1 and B_2, respectively;
4. remove edge e and add edges e_1 and e_2 in the edge list such that the two pointers of e_1 point to u in B_1 and v in B_2, and the two pointers of e_2 point to v in B_1 and w in B_2.

Fig. 11. SUBDIV algorithm

of leaves in the tree [33], and that traversing from any leaf to the root in a 2-3 tree, inserting a new node, deleting a node, splitting a 2-3 tree into two smaller trees, and concatenating two 2-3 trees into a larger tree all take logarithmic time [33].

The COFACIAL algorithm is implemented by two traversings from leaves to roots in the 2-3 trees in the DLFL structure, the INSERT, DELETE, and SUBDIV algorithms are implemented by a small number of insertions, deletions, splittings, and concatenations on the 2-3 trees in the DLFL structure. Therefore, we conclude that all these algorithms take logarithmic time under the DLFL structure. □.

3. User Interface for Topological Operations

INSERT and DELETE operations in *internal representation* may change the topology of a mesh. INSERT operation requires a special attention. An inserted edge must satisfy some constraints to get a unique topological description. To explain why we need some constraints, we need to look at rotation systems more closely. A rotation system is defined by the order of connections for every edge. In other words, from the 3-dimensional positions and orientations of the edges, it should be possible to get a unique order of connection to update the rotation system. If the user draw any edge freely in 3-dimensional space, it is not possible to give a unique order. Therefore, in such a situation, ambiguities can occur and edge insertion may not correspond to one unique topology as shown in Figure 12. In other words, we need to ensure that the user defines a unique order for the connections between vertices.

The example we have shown in Figure 3 illustrates a constraint that can be used: the new inserted edge must be drawn on the surface of 2-manifold in which the mesh belongs. In order to enforce such a constraint we must force the user to draw each new edge over the 2-manifold. As it can easily be seen, this constraint is too restrictive for interactive modeling. Fortunately, this constraint is sufficient but not necessary to get a unique topological description. There exists another simpler constraint that is both sufficient and necessary.

To illustrate this simple constraint, let us look at the property of rotation systems. Since, for any 2-manifold, a small neighborhood around every point is homeomorphic to an open disk, it is possible to define a disk around every vertex of

the mesh as shown in Figure 13. If every edge originated from a given vertex is constrained to the disk of this vertex, it is possible to give an order. For example as shown in Figure 13 in a planar disk that includes a given vertex, it is possible to determine the order of connections. In other words, if the inserted edges should be constrained to the disks that are in 2-manifold surface and include the end vertices of the edges, it is possible to determine a unique rotation system and a unique topology. This is not extremely restrictive constraint. Each vertex can be defined by a point and a tangent plane which are shown as a space circle. Each edge leaving a vertex can be constrained to the plane. The user can change the order of the connection and orientation of the tangent plane. Note that these edges need to be represented at least a third degree parametric curve such as Bezier or Hermitian curve [1].

The minimum constraint explained above suggests that the topological operations can be effectively done even in 2-dimensional space. To provide an effective interface for handling 2-dimensional manifolds, we also propose a new visual representation that will be used as an interface in the modeling topology of 2-dimensional manifolds. This 2-dimensional visual representation is useful since it can clearly separate topological and geometrical operations.

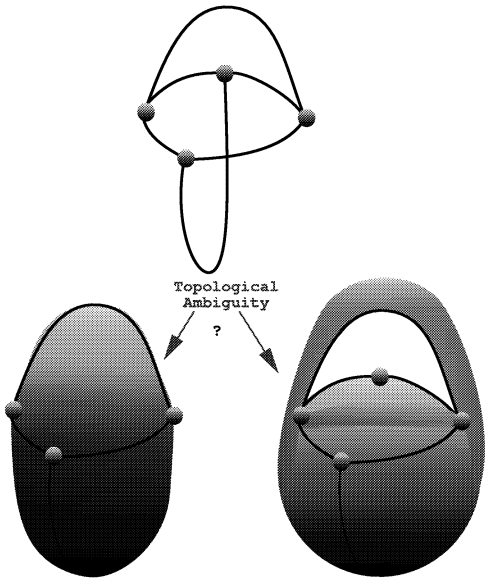

Fig. 12. An edge insertion without any constraint may not define one unique topology.

4. Visual Representation of Topology and 2D User-Interface

It is well-known in topology that the orientable 2-dimensional manifold of genus n can be represented by a $4n$ sided polygon, called its *polygonal representation* [25].

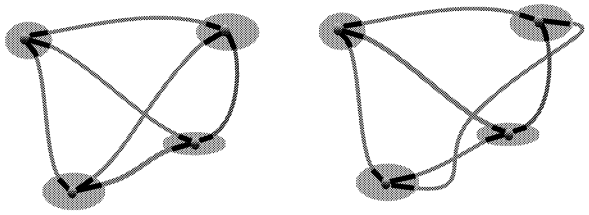

Fig. 13. A simple constraint around vertices is enough to define unique topology.

The $4n$ sided polygon ($4n$-gon) that corresponds to the 2-dimensional manifold with genus n can be obtained by $2n$ cuts on the manifold. For each handle, we need to make 2 cuts. For example, the torus in Figure 2 can be represented in its polygonal representation in the rectangle shown in Figure 14, with the opposite sides of the rectangle being identified. Using the techniques we developed for DLFL, we can also apply topological operations efficiently to the polygonal representations of 2-dimensional manifolds. Figure 15 shows the process of obtaining a polygonal representation going from a $4n$-gon with one handle to a $4(n+1)$-gon.

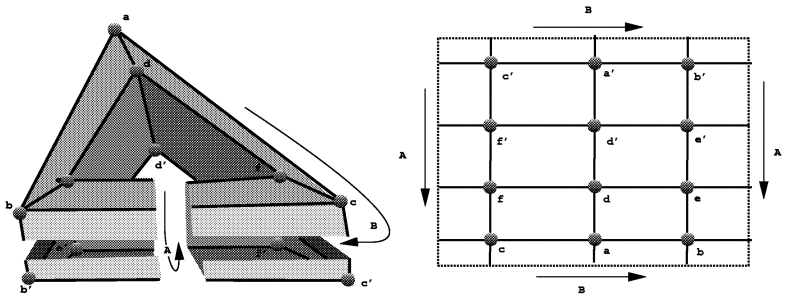

Fig. 14. Polygonal representation of previous graph G.

The polygonal representation of a manifold can be obtained efficiently using the method of graph embeddings based on our DLFL structure. In order to obtain the polygonal representation of an orientable 2-dimensional manifold of genus k, we first embed a bouquet of $2k$ edges into the manifold (a *bouquet* of $2k$ edges is a single-vertex graph with $2k$ selfloops) so that the embedding has only a single face [34], then cut the manifold along the edges of the bouquet to obtain a $4k$ polygonal representation of the manifold. Note that this $4k$ sided polygon is actually the face boundary walk of the unique face for the embedded bouquet of $2k$ edges. Therefore, the $4k$ sided polygon representation can be easily obtained from the DLFL representation of the embedded bouquet. Moreover, suppose we add a new handle to a 2-dimensional manifold given in its polygonal representation, the polygonal representation for the new 2-dimensional manifold can be easily obtained by inserting

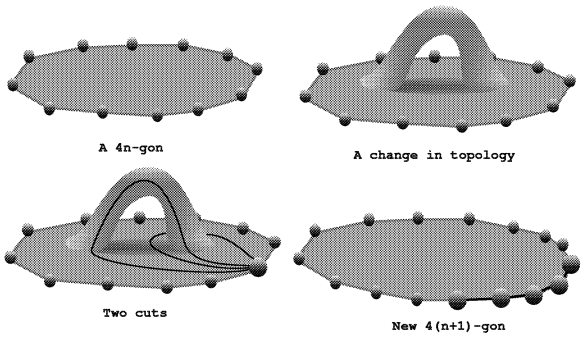

Fig. 15. Obtaining $4(n+1)$ sided polygon.

two new edges (see Figure 15 for an illustration). Similarly, removing a handle in the 2-dimensional manifold can be easily implemented by deleting two edges in the bouquet. In summary, the polygonal representation of a 2-dimensional manifold can be conveniently managed by the DLFL structure on a one-face embedding of the bouquet.

The polygonal representation of a 2-dimensional manifold has the advantage that the figures drawn on a local area in the manifold look exactly the same as on its polygonal representation. Therefore, local manipulations on a manifold can be performed conveniently and intuitively on the corresponding polygonal representation. For local manipulations interesting figures need to be moved to the center of the polygon for easier manipulations. Moving in a polygonal representation can be performed effectively by using the fact that all the corners of the polygonal representation correspond to the same point in the 3-dimensional space. In other words, in order to move in a polygonal representation, it is possible to cut the 2-dimensional manifold surface differently. Since it is possible to move the cutting edges slowly, the user can perceive the change as a move in a polygonal representation as shown in Figure 16.

An additional advantage of the polygonal representation is that the positions of the vertices in the $4n$ sided polygon can be used as the texture coordinates for the mesh surface and the users can control texture mapping by changing the vertex positions in the polygonal representation.

The polygonal representation provides us a paradigm for an interface for topology control. The interface can be similar to 2-dimensional graph drawing or drawing systems. By using such an interface, the user can make manipulations over the topology and change texture coordinates. The user is also able to define and merge polygonal representations of several meshes.

5. Discussion and Future Work

Our theoretical framework does not take into account the 3-dimensional posi-

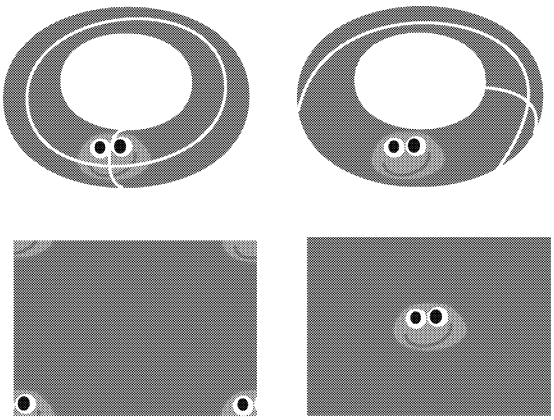

Fig. 16. Moving a figure to the center by changing cutting edges.

tions of the vertices. Some choices of these positions may result in self-intersection of the meshes. Such a self-intersection is not desirable unless the user particularly wants to achieve it.

The problem with self-intersections can easily be explained with 1-dimensional manifolds. As we know a polygon in 2-dimensional space is a 1-dimensional manifold. Therefore, we expect that a polygon separates the 2-dimensional space into three regions: inside the polygon and outside the polygon and the polygon itself. However, by moving a vertex in 2-dimensional space, the user can create a self-intersecting polygon which has neither inside nor outside. In 2-dimensional drawing systems, this is not considered a problem since the user can easily correct it. If the user does not correct a self-intersection, the result can still be considered to be an acceptable shape and in some systems, such self-intersecting polygons are filled by using geometry-based inside-outside tests. This approach to self-intersection used in 2-dimensional drawing systems is not useful for modeling meshes. Since some of the faces of the mesh will be hidden, the user can create self-intersection without even realizing it. Moreover, unlike the 2-dimensional case, it is not easy for the user to detect such self-intersection. Therefore, if the user does not want to create self-intersection, the system must prevent it from occuring by restricting the positions of the vertices. We are currently working on several possible solutions to this problem.

The DLFL structure has been implemented as a C++ class. We are also planning to develop a Java version for the structure.

6. Conclusions

We have proposed a conceptual framework for the development of systems for modeling topologically consistent meshes. We combine the graph rotation system representation and the polygonal representation into one integrated system, in which the graph rotation system is used as an internal representation for manifolds, while the polygonal representation is used in the user interface. We also provide efficient topological operations for edge insertion and deletion, and vertex and polygon tracing.

Acknowledgements

Authors are thankful to Donald H. House and reviewers for their helpful suggestions. Ergun Akleman's work is supported in part by the Texas A&M, Scholarly & Creative Activities Program. Jianer Chen's work is supported in part by the National Science Foundation under Grant CCR-9613805.

References

1. R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, (Morgan Kaufmann Publishers, Los Altos, CA, 1987).
2. C. Loop and T. D. Rosa, "Generalized B-spline surfaces with arbitrary topology", *Computer Graphics*, **24** (August 1991) 101-165.
3. C. Loop, "Smooth spline surfaces over irregular meshes", *Computer Graphics*, **28** (August 1994) 303-310.
4. D. Doo and M. Sabin, "Behaviour of recursive subdivision surfaces near extraordinary points", *Computer Aided Design*, **10** (September 1978) 356-360.
5. E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes", *Computer Aided Design*, **10** (September 1978) 350-355.
6. M. Halstead, M. Kass, and T. DeRose, "Efficient, fair interpolation using Catmull-Clark surfaces", *Computer Graphics*, **27** (August 1993) 35-44.
7. C. Loop, "Smooth subdivision surfaces based on triangles", Master's Thesis, Department of Mathematics, University of Utah (1987).
8. G. Barequet and S. Kumar, "Repairing CAD models", in *Proceedings of IEEE Visualization'97*, (October 1997) pp. 363-370.
9. 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.
10. K. Weiler "Edge-based data structures for solid modeling in curved-surface environments ", *IEEE Computer Graphics and Applications*, (January 1985) 21-40.
11. B. J. Baumgart, "Winged-edge polyhedron representation", Technical Report CS-320, Stanford University, 1972.
12. K. Weiler "Polygon comparison using a graph representation", *Computer Graphics*, **13** (August 1980) 10-18.
13. B. J. Baumgart, "A polyhedron representation for computer vision", in *44th AFIPS National Computer Conference*, (1975) pp. 589-596.
14. L. Guibas, J. Stolfi, "Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams", *ACM Transaction on Graphics*, **4** (1985) 74-123.
15. C. M. Hoffmann, *Geometric & Solid Modeling, An Introduction*, (Morgan Kaufman

Publishers, Inc., San Mateo, Ca., 1989).
16. C. M. Hoffmann and G. Vanecek, "Fundamental techniques for geometric and solid modeling", *Manufacturing and Automation Systems: Techniques and Technologies*, **48** (1990) 347-356.
17. A. Ricci, "A constructive geometry for computer graphics", *The Computer Journal*, **16** (May 1973) 157-160.
18. A. A. G. Requicha and H.B. Voelcker, "Solid modeling: a historical summary and contemparay assessment", *IEEE Computer Graphics and Applications*, **2** (March 1982) 9-24.
19. M. Mantyla, "Boolean operations on 2-manifolds through vertex neighborhood classification", *ACM Transaction on Graphics*, **5** (January 1986) 1-29.
20. K. Weiler, "Topological structures for geometric modeling", Ph.D. Thesis, Rensselaer Polytechnic Institute, N. Y., August 1986.
21. M. Karasick, "On the representation and manipulation of rigid solids", Ph.D. Thesis, McGill University, 1988.
22. . G. Vanecek, "Set operations on polyhedra using decomposition methods", Ph.D. Thesis, University of Maryland, 1989.
23. A. T. Fomenko and T. L. Kunii, *Topological Modeling for Visualization*, (Springer-Verlag, New York, 1997).
24. S. Takahashi, Y. Shinagawa and T. L. Kunii, "A feature-based approach for smooth surfaces", in *Proceedings of Fourth Symposium on Solid Modeling*, (1997) pp. 97-110.
25. P. A. Firby and C. F. Gardiner, *Surface Topology*, (John Wiley & Sons, 1982).
26. J. L. Gross and T. W. Tucker, *Topological Graph Theory*, (Wiley Interscience, New York, 1987).
27. L. Heffter, "Uber das problem der nachbargebiete", *Math. Annalen*, **38** (1891) 477-508.
28. J. Edmonds, "A combinatorial representation for polyhedral surfaces", *Notices American Mathematics Society*, **7** (1960) 646.
29. J. Chen, D. Archdeacon, and J. Gross, "Maximum genus and connectivity", *Discrete Mathematics*, **149** (1996) 19-29.
30. J. Chen, J. Gross and R. Riper, "Overlap matrices and imbedding distributions", *Discrete Mathematics*, **128** (1994) 73-94.
31. J. Chen, "Algorithmic graph embeddings", *Theoretical Computer Science*, **181** (1997) 247-266.
32. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, (Springer-Verlag, 1985).
33. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading, MA, 1974).
34. J. Chen and J. Gross, "Limit points for average genus I. 3-connected and 2-connected simplicial graphs", *J. Combinatorial Theory Ser. B*, **55** (1992) 83-103.