Topologically Robust Mesh Modeling: Concepts, Data Structures and Operations

JIANER CHEN
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
chen@cs.tamu.edu

ERGUN AKLEMAN
Visualization Laboratory
Texas A&M University
College Station, TX 77843-3137
ergun@viz.tamu.edu

Abstract

We extend the theory of graph rotation systems and provide a solid foundation for 2-manifold mesh modeling. Based on this theory, we identify a group of simple validity rules, and show that the validity of 2-manifold structures can be tested very efficiently on all existing data structures for mesh modeling. Moreover, the new theory enables us to develop very efficient implementations for manifold preserving operations, which are essential for a robust interactive modeling system. Our results can be adopted by any existing modeling softwares without a major change.

Keywords. *mesh modeling, topological robustness, topological graph theory*

1 Introduction

Modeling 2-manifold meshes with a simple user interface has been an important issue in computer graphics and computer aided geometric design. Modeling with 2-manifold meshes has its obvious advantages, including: (1) dealing with non-manifold structures greatly complicates modeling algorithms [16, 18]; and (2) many subdivision algorithms [27] specifically require 2-manifold mesh structures. Moreover, non-manifold looking geometric shapes can be effectively represented using 2-manifold mesh structures [15].

Although the importance of 2-manifolds is widely recognized in modeling community [14, 16], most commercial modeling systems allow non-manifold structures, because of the following reasons: (1) certain popular modeling operations, such as set operations, can easily introduce non-manifold structures; and (2) users may intentionally want to create non-manifold structures. (See supplemental materials for an example.)

This flexibility of the commercial modelers can be considered an advantage for novice users, but it eventually becomes a hurdle for experienced users in modeling shapes with complicated topology. For instance, changing topology to create a very high genus surface requires a great deal of work in such commercial systems. Moreover, certain important and popular modeling operations that specifically require 2-manifold structures now become unreliable. For example, MAYA may crash when a subdivision operation is applied on a non-manifold shape that was created unintentionally.

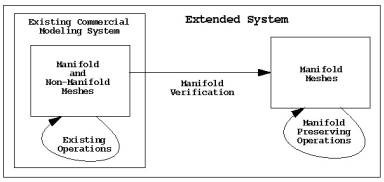


Figure 1. An extended modeling system

Since complete re-design of the existing systems will be extremely expensive and difficult, a solution for solving the above problem is to augment the systems with a 2-manifold modeling environment, as shown in Figure 1. Such an extension will keep the flexibility of the current modeling systems while providing high efficiency and reliability for modeling processes dealing with 2-manifold shapes of complicated topological structures.

Such an extension is involved in two important issues (see Figure 1): (1) effective verifications of 2-manifold structures, which is necessary to make sure that the shapes passed to the 2-manifold modeler are valid 2-manifold structures; and (2) manifold preserving operations, which are employed by the 2-manifold modeler so that the 2-manifold modeler can deal with all 2-manifold structures but will never create non-manifold structures. We point out that these issues have also been important in the research of mesh simplifications (see, for example [12]).

The problem of verifying 2-manifold validity of a given mesh structure has been studied by a number of researchers. Hoffmann presented an algorithm [16] for verification of the 2-manifold validity of triangulated mesh structures. Guibas

and Stolfi gave a set of rules for deciding whether an edge algebra describes a valid 2-manifold structure [14]. Unfortunately, all previous approaches have their limitations. For example, to apply Hoffmann's algorithm to a general mesh, the mesh must be first triangulated, while to apply Guibas and Stolfi's algorithm, which is applicable to general mesh structures, a mesh must be first converted into the edge algebra representation (such as the quad-edge structure [14]). The study of manifold preserving operations has been closely related to that for 2-manifold validity verification [12, 14, 18], which has been centered on the completeness (i.e., the ability of creating all 2-manifold mesh structures), soundness (i.e., the robustness of not generating non-manifold structures), and efficiency.

Our main contribution in the current paper is to establish a universal framework for the theory and practice of 2-manifold modeling systems. Based on the classical theory of graph rotation systems studied in topological graph theory, we first build a solid theoretical foundation for 2-manifold modeling. Our theoretical work enables us to give very simple rules for testing topological validity of mesh structures. Our validity rules are universal in the sense that they are independent of mesh structures and representations. We also study a set of manifold preserving operations that can be easily implemented in any existing system. Theory, data structures, and algorithms related to our new framework are thoroughly discussed. More specifically, our contributions include:

- We establish a solid theoretical foundation for topologically robust mesh modeling, by extending the classical theory of graph rotation systems;
- Based on the established theoretical foundation, we propose new testing rules for topological validity of mesh modeling: a mesh structure represents a valid 2-manifold if and only if (1) each edge in the mesh induces exactly two directed edges; and (2) each vertex in the mesh is uniquely associated with a cyclically ordered list containing all directed edges from the vertex;
- We present efficient topological validity testing algorithms based on various modeling data structures and representations that have been widely used in mesh modeling. This demonstrates that the new testing rules for topological validity are universal and can be easily incorporated in any existing modeling systems;
- We propose new data structures for topologically robust mesh modeling, develop efficient modeling algorithms on the new data structures, and compare the new data structures and algorithms with those employed in existing modeling systems;
- We study the set of modeling operators proposed in our earlier work [5], and show that the operators can be easily implemented for any mesh representation. We prove that the operator set can create all 2-manifold meshes but never result in non-manifold structures. The operator set is simple, intuitive, and more effective, from the viewpoints of both theoretical concepts and practical implementations.

In conclusion, our research provides strong evidence for the feasibility of augmenting the existing modeling systems with a 2-manifold modeler, without involvement of any major changes, which will greatly enhance the efficiency and reliability of the current modeling technology.

2 Preliminaries

A *2-manifold* is a topological space where every point has a neighborhood homeomorphic to an open disk. A 2-manifold is *orientable* if it does not contain a Möbius band [13]. In this paper, we will assume, unless explicitly stated, that all 2-manifolds in our discussion are orientable 2-manifolds. A connected 2-manifold is called a *surface*. Thus, a 2-manifold in general consists of a number of surfaces. An example of a surface is a sphere S in the 3-dimensional Euclidean space E^3. By Euler's Characteristic Theorem [13], each surface is homeomorphic to the sphere S with zero or more handles. The number of handles on the sphere S is called the *genus* of the surface. The *genus* of a 2-manifold is the sum of genera of its component surfaces.

A *mesh* on a 2-manifold S specifies an embedding $\rho(G)$ of a graph G on the 2-manifold S. Each connected component of $S-\rho(G)$ plus the bounding edges in G makes a *face* of the mesh. Following the conventions adopted by many other researchers [15, 19, 26], we allow a mesh to have multiple edges (i.e., more than one edge connecting the same pair of vertices) and self-loops (i.e., edges with both ends at the same vertex). The mesh is *cellular* if the interior of each face is homeomorphic to an open disk. Non-cellular meshes have also been studied in the literature but the corresponding modeling algorithms are more complicated [16, 19]. In this paper, we only consider cellular meshes.

Let G be a mesh on a 2-manifold S. Because each vertex v of G has a neighborhood in S homeomorphic to an open disk, the edges incident on v are cyclically ordered in this neighborhood. This observation has turned out to be extremely important in representing a 2-manifold mesh. The classical *winged-edge data structure* proposed by Baumgart [7] for mesh representations was essentially developed based on this observation.(See supplemental materials for winged-edge structure.)

Thus, the winged-edge structure is *edge-based*, which describes the mesh structure in terms of the adjacency information on each edge of the mesh. An adaptation of the winged-edge structure is the *half-edge* data structure [19],

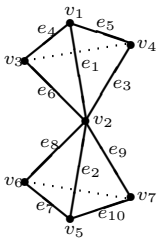
which is a hierarchical data structure. The major difference between the half-edge data structure and other structures is the introduction of the concept of *half-edges*[1]. A half-edge describes "one-side" of an edge of the mesh, thus is uniquely associated with a face of the mesh. A half-edge node consists of references to the face, to its starting vertex, to the corresponding edge, and to the previous and the next half-edges of the face. The half-edge data structure indicates very clearly a property of the mesh on a 2-manifold that each edge of the mesh appears exactly twice on the boundaries of the faces of the mesh.

Guibas and Stolfi [14] studied mesh structures by defining an algebra on the edges of a mesh and its dual (thus, it works specifically for cellular meshes). Each edge can be given a *direction* along which the edge is traversed, and an *orientation* by which the "left side" and "right side" of the edge are well-defined when traversing the edge along the given direction on the 2-manifold. Thus, each edge has four different directed and oriented versions. Each directed and oriented edge has four attributes: the two end-vertices *Org* and *Dest* (equivalent to *vstart* and *vend* in the winged-edge structure), and the two faces *Right* and *Left* on the two sides of the edge on the 2-manifold (equivalent to *fcw* and *fccw* in the winged-edge structure). An *edge algebra* is introduced by defining three functions *Flip*, *Onext*, and *Rot* on the set of the directed and oriented edges, where *Flip* flips the orientation of an edge, *Onext* gives the "next" edge of an edge based on its *Org* (thus is similar to *nccw* in the winged-edge structure); and *Rot* "rotates" the edge (which is a little more technical and we omit its detailed explanation). Both orientable and non-orientable 2-manifolds can be described by edge algebras. Moreover, an edge algebra gives the structures for both the primary mesh and its dual. A data structure, the *quad-edge* structure, has been introduced to represent the edge algebra for a mesh structure [14].

When concentrating on orientable 2-manifolds, as most modeling applications do, the edge algebra and its corresponding quad-edge structure can be simplified. Since all edges of a mesh on an orientable 2-manifold have consistent orientations, the edge orientations need not be given explicitly. Thus, each edge of the mesh has only two different directed versions. Again three functions *Onext*, *Sym*, and *Dual* are needed to define an edge algebra on the edge set of a mesh. In this case, each edge node in the corresponding quad-edge structure contains four pointers to essentially the four edge nodes corresponding to the edge nodes *ncw*, *pcw*, *pccw*, and *nccw* in the winged-edge structure.

For manifold preserving operations, Mäntylä [19] proposed Euler operators, and Guibas and Stolfi [14] defined the splice and create-edge operators. The completeness, soundness, and efficient implementations of these operators have been thoroughly studied.

[1]There are a number of variations and renamings for the concept of half-edges adopted in commercial modeling systems, such as "co-edges" in ACIS (Spatial Technologies Inc.) [22] and "fin" in Parasolid (Unigraphics Solutions Inc.) [25].



name	*vstart*	*vend*	*fcw*	*fccw*	*ncw*	*pcw*	*nccw*	*pccw*
e_1	v_1	v_2	f_1	f_2	e_6	e_4	e_5	e_3
e_2	v_2	v_5	f_3	f_4	e_7	e_8	e_9	e_{10}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

where the faces are: $f_1 = \{v_1 v_3 v_2\}$, $f_2 = \{v_1 v_2 v_4\}$, $f_3 = \{v_2 v_6 v_5\}$, $f_4 = \{v_2 v_5 v_7\}$.

Figure 2. A non-manifold under winged-edge structure

3 Testing topological validity

We observe that most existing mesh representations and data structures may accept non-manifold structures. For example, the left shape in Figure 2 is not a 2-manifold (the point v_2 does not have a neighborhood homeomorphic to an open disk). However, the right table in the figure is a valid winged-edge data structure for the corresponding mesh. It is not difficult to verify that other data structures, such as the half-edge data structure and quad-edge data structure, can also describe the mesh structure in Figure 2.

As we have discussed earlier, there has no universal solution yet for the verification of 2-manifold validity for mesh structures. Existing methods either require a special mesh representation [14] or are only applicable to meshes of special structures [16]. In this section, we propose a new approach for verification of 2-manifold validity, by extending the classical theory of rotation systems studied in topological graph theory [13]. We show that this approach can be adopted by all existing representations and data structures for verification of 2-manifold validity of mesh structures.

Let G be a mesh on a 2-manifold S. First note that since the mesh is cellular, each connected component of the graph G corresponds to a unique surface in S [14]. Since each point on S has a neighborhood homeomorphic to an open disk, if one stands at a vertex v on S, then he can see in a small neighborhood of v on S that the edges from v are cyclically ordered. Thus, two consecutive edges from v form a *face corner* for a face of the mesh. This induces two important concepts: (1) each edge incident on v is given a "direction" starting from v (note that a self-loop at v then induces two directed edges of opposite directions from v); and (2) these directed edges from v form a natural "rotation". If v is an isolated vertex without any incident edges, then since the mesh G is cellular, the surface containing v

must be a genus 0 sphere S_2. Note that $S_2 - \{v\}$ is homeomorphic to an open disk whose boundary degenerates into a single point v, i.e., v is a valid cellular mesh on the sphere S_2. The sphere S_2 will be called a *point-sphere* containing the vertex v. Since the isolated vertex v has no incident edges, we define that v has a *empty rotation* that is an empty list of directed edges.

Thus, the mesh G on the 2-manifold S induces a "rotation system" on the corresponding graph G, in which each vertex is given a unique rotation of directed edges. Based on this observation, we introduce the following important concept.

Definition Let G be a graph of n vertices. Each edge in G induces two *directed edges* of opposite directions. A *rotation* at a vertex v of G is a cyclic ordering of the directed edges from v (an isolated vertex without incident edges always has the empty rotation). A *rotation system* of the graph G consists of a set of n rotations, each for a vertex in G.

Remark 1. The above definition is an extension of the same concept defined on graphs without isolated vertices. The original concept of graph rotation systems on graphs without isolated vertices has been extensively used in topological graph theory [13]. We will see that the extension of graph rotation systems to including isolated vertices without incident edges is fundamentally important in the study of 2-manifold mesh modeling.

Remark 2. Note that in the definition of graph rotation systems, the graph G is not necessarily connected. Moreover, multiple edges and self-loops are also allowed. Of course, in case of multiple edges, the edges with the same ends must be distinguishable, and in case of self-loops, the two directions of the self-loop must be specified.

Remark 3. If the two end-vertices u and w of an edge $[u, w]$ are different, then the induced directed edges can be simply given by $\langle u, w\rangle$ and $\langle w, u\rangle$. We will put a " ˆ " above a symbol such as $\hat{e}$ to refer to a directed edge. For a directed edge $\hat{e}$, we also denote by $r(\hat{e})$ the corresponding reversed directed edge. Therefore, if $\hat{e} = \langle u, w\rangle$, then $r(\hat{e}) = \langle w, u\rangle$.

For graphs without isolated vertices, it has been proved [11] that a graph rotation system uniquely determines an embedding of the graph on a 2-manifold. We extend this important result to include the cases of isolated vertices. Moreover, our proofs are more constructive and more algorithmic, which clearly illustrate how this new structure is used in 2-manifold modeling. We state the main result here without proof and refer interested readers to [9] for the formal and detailed discussions on these theoretical results.

Theorem 3.1 *A mesh structure defines a valid 2-manifold if and only if it uniquely specifies a graph rotation system $\rho(G)$, whose face set is constructed by the algorithm* **Facing**$(\rho(G))$ *given in Figure 3.*

Subroutine FaceTrace($\langle u_0, w_0\rangle$) {* $\langle u_0, w_0\rangle$ is a directed edge *}

1. trace $\langle u_0, w_0\rangle$;
2. let $\langle u, w\rangle$ be the directed edge next to $\langle w_0, u_0\rangle$ in the rotation at w_0, where $u = w_0$;
3. **while** ($\langle u, w\rangle \neq \langle u_0, w_0\rangle$) **do**
   let $\langle w, x\rangle$ be the directed edge next to $\langle w, u\rangle$ in the rotation at w;
   trace $\langle u, w\rangle$; $\langle u, w\rangle = \langle w, x\rangle$.

Algorithm Facing ($\rho(G)$) {* $\rho(G)$ is a graph rotation system *}

1. **for** each isolated vertex v without incident edges in G **do**
   construct a face with a degenerated boundary v;
2. **while** there is a directed edge $\langle u_0, w_0\rangle$ not traced yet **do** FaceTrace($\langle u_0, w_0\rangle$).

Figure 3. Face reconstruction based on graph rotation systems

4 Data structures and topological validity

Graph rotation systems are *vertex-based* descriptions of mesh structures. Theorem 3.1 shows that a valid graph rotation system guarantees the validity of the underlying 2-manifold mesh structure. In this section, we show that based on Theorem 3.1 it is in general trivial to check whether a mesh structure gives a valid 2-manifold based on any existing mesh representations.

To simplify the discussion, we will assume from now on that our graphs do not contain multiple edges and self-loops. Thus, edges and directed edges now can be uniquely specified by pairs of vertices and ordered pairs of vertices, respectively (The discussion can be extended to graphs with multiple edges and self-loops without much technical troubles except using more tedious notations.) According to Theorem 3.1, we have the following simple rules to check whether a mesh structure represents a valid 2-manifold:

> **Validity Rules.** A mesh structure gives a valid 2-manifold if and only if
>
> 1. Each edge $[u, w]$ induces exactly two directed edges $\langle u, w\rangle$ and $\langle w, u\rangle$;
> 2. For each vertex v, a unique cyclically ordered list is specified that contains all and only the directed edges from v.

It is easy to see that the Validity Rules above are consistent with those proposed by Hoffmann [15] when a mesh is triangulated, which claims that a triangulated mesh represents a valid 2-manifold if every edge is shared by exactly two triangles and each vertex is uniquely associated with a ring of triangles incident on it. The advantage of our Validity Rules is that we do not restrict the meshes to be

triangulated. Moreover, our rules seem simpler in the sense that the rules are based on 1-dimensional cells (i.e., directed edges) while Hoffmann's rules are based on 2-dimensional cells (i.e., triangles).

We point out that test rules similar to what we propose here may have been used in modeling practice. However, to our knowledge, this is the first time that these rules are formally stated and mathematically proven. These theoretical results grant the first time the validity of using these test rules in 2-manifold modeling.

In the rest of this section, we demonstrate another advantage of our validity rules, by showing that our rules are directly applicable to any existing mesh representations.

4.1 Vertex-based data structures

There is a fairly natural data structure for representation of graph rotation systems, the *graph adjacency list* structure, which is the most commonly used data structure for graphs in computer science. Recall that an adjacency list for a graph G is given by an array $A[1..n]$ of linked lists, where $A[v]$ is a linked list that contains all neighbors of the vertex v. Since the graph has no multiple edges and self-loops, each vertex w in the list $A[v]$ uniquely specifies the directed edge $\langle v, w\rangle$ from the vertex v. Since for each edge $[v, w]$, the vertex w appears in the list $A[v]$ and the vertex v appears in the list $A[w]$, Validity Rule 1 is satisfied. Moreover, if we interpret for each vertex v the list $A[v]$ as the cyclic ordering of the directed edges from v, then the list $A[v]$ specifies a unique rotation of the directed edges from the vertex v. Thus, Validity Rule 2 is satisfied. Thus, we conclude

Theorem 4.1 *A valid graph adjacency list $A[1..n]$ represents a valid 2-manifold structure.*

The drawback of graph adjacency lists for mesh modeling is its time complexity for supporting modeling operations. For example, when we trace a face boundary following a directed edge $\langle v, w\rangle$, we need to jump from the list $A[v]$ to the list $A[w]$ and find the directed edge following the directed edge $\langle w, v\rangle$ in the list $A[w]$. The time for doing this is proportional to the size of the list $A[w]$, which will be time consuming in particular when the vertex w has a large valence.

4.2 Edge-based data structures

Most data structures in mesh modeling are edge-based. We will focus our discussion on the well-known winged-edge structure, then extend our discussion to other structures.

Recall that each edge node in a winged-edge structure consists of nine components: (*name, vstart, vend, fcw, fccw, ncw, pcw, nccw, pccw*) (see Figure 9). By the definitions, the edge $[$*vend*$, $*vstart*$]$, and vertex *vend*, and the edge *ncw* form a corner of the face *fcw*, and the edge $[$*vstart*$, $*vend*$]$, the vertex *vstart*, and the edge *nccw* form a corner of the face *fccw*. Thus, if the mesh structure represents a valid 2-manifold, then the edges *nccw* and *ncw* should induce the "next" directed edges from the vertices *vstart* and *vend*, respectively, for the directed edges $\langle$*vstart*$, $*vend*$\rangle$ and $\langle$*vend*$, $*vstart*$\rangle$. Thus, whether the winged-edge structure gives a valid 2-manifold can be verified based on the components *ncw* and *nccw*.

Denote by $A[e]$ the edge node for the edge e in the winged-edge structure. By $A[e]$.*vstart*, $A[e]$.*vend*, $A[e]$.*ncw*, and $A[e]$.*nccw*, we refer to the corresponding components *vstart*, *vend*, *ncw*, and *nccw* in $A[e]$. Consider the algorithm in Figure 4.

Subroutine VertexTrace(v_0, e_0) {* the vertex v_0 is an end of the edge e_0 *}

1. **if** ($A[e_0]$.*vstart* $= v_0$) **then** $e = A[e_0]$.*nccw* **else** $e = A[e_0]$.*ncw*;
2. **while** ($e \neq e_0$) **do**
2.1. suppose $e = [v_0, w]$;
2.2. **if** (the directed edge $\langle v_0, w\rangle$ has been marked) **then** Stop('not manifold');
2.3. mark the directed edge $\langle v_0, w\rangle$;
2.4. **if** ($A[e]$.*vstart* $= v_0$) **then** $e = A[e]$.*nccw* **else** $e = A[e]$.*ncw*;
3. suppose $e_0 = [v_0, w_0]$; mark the directed edge $\langle v_0, w_0\rangle$.

Algorithm E-Validity ($A[1..m]$) {* $A[1..m]$ is a winged-edge structure *}

for each edge node $A[e]$ **do**
1. $u = A[e]$.*vstart*; $w = A[e]$.*vend*;
2. **if** (u is not marked) { **then** mark u; VertexTrace(u, e); }
else if (the directed edge $\langle u, w\rangle$ has not been marked) **then** Stop('not manifold');
3. **if** (w is not marked) **then** { mark w; VertexTrace(w, e); }
else if (the directed edge $\langle w, u\rangle$ has not been marked) **then** Stop('not manifold').

Figure 4. Topological validity testing on winged-edge structure

Theorem 4.2 *If a winged-edge structure can pass the test of the algorithm* **E-Validity**, *then it represents a valid 2-manifold. Moreover, the test algorithm runs in time $O(m)$.*

We give a brief explanation for the correctness of the algorithm **E-Validity**, and refer interested readers to [9] for a formal proof. First note that the vertices are marked in the main algorithm **E-Validity** while directed edges are all marked in the executions of the subroutine **VertexTrace**. Thus, a call to **VertexTrace** on a vertex v_0 checks the condition that all directed edges from v_0 must form a unique rotation at v_0, while steps 2 and 3 in the main algorithm **E-Validity** verify that each edge in the mesh induces exactly two directed edges.

Remark. Theorem 4.2 indicates that the original winged-edge structure proposed by Baumgart [7], in which

only the components *name*, *vstart*, *vend*, *ncw*, and *nccw* are included, will be sufficient for modeling 2-manifold mesh structures.

As an example, we show how the algorithm **E-Validity** reports an error on the winged-edge structure given in Figure 2. By looking at the edge node $A[e_1]$, step 3 of the algorithm will mark the vertex v_2 and call the subroutine **VertexTrace**(v_2, e_1). The subroutine **VertexTrace**(v_2, e_1) will mark the directed edges $\langle v_2, v_3\rangle$, $\langle v_2, v_4\rangle$, and $\langle v_2, v_1\rangle$ and return back to the main algorithm **E-Validity**. Then the main algorithm **E-Validity** looks at the edge node $A[e_2]$ and finds out that the vertex v_2 is marked while the directed edge $\langle v_2, v_5\rangle$ is not marked, thus correctly reports an error.

We briefly describe the Validity Rules on other edge-based data structures.

In the half-edge structure [19], a half-edge corresponds uniquely to a directed edge in our rotation system. Moreover, each half-edge has a unique reference to the next half-edge of the corresponding face, from which we can easily retrieve the information for the next directed edge based on a vertex. Thus, the verification of the Validity Rules can be done by checking whether the reference to the next half-edge of each half-edge induces a valid rotation system. It is not difficult to verify that testing the Validity Rules on a half-edge structure can be done in time $O(m)$, where m is the number of edges in the mesh.

In the quad-edge structure [14] for an edge algebra for an orientable 2-manifold, each edge has two directed versions, corresponding to the two directed edges in our graph rotation systems. The operation *Onext* defined in the edge algebra gives exactly the next directed edge of a given directed edge from a vertex. Thus, again, based on the directed edges and the *Onext* operation, we can conveniently verify whether the quad-edge structure gives a valid graph rotation system thus represents a valid 2-manifold structure. Again the Validity Rules on the quad-edge structure can be tested in time $O(m)$, where m is the number of edges. Note in particular that our validity test on quad-edge structure does not need the information for the dual mesh structure, which is also provided in the quad-edge structure and is required for the validity test using a group of rules proposed in the original research for edge algebra [14].

4.3 Face-based data structures

Face-based data structures have been regarded as the least reliable for modeling 2-manifold structures. In particular, a variety of non-manifold structures, such as wrongly-oriented polygons, overlapping polygons, missing polygons, cracks, and T-junctions, can be resulted from a face-based data structure [21].

We have recently re-investigated face-based data structures for mesh modeling and proposed a new face-based data structure [2]. The advantage of this new data structure is its extremely efficient support to a variety of modeling operations. The new data structure has been implemented in our 2-manifold modeling system [3, 4]. In this subsection, we show how to test the Validity Rules on this face-based data structure.

VertexTrace$(v_0, \langle v_0, w_0\rangle)$

1. let $\langle v_0, x\rangle$ be the directed edge following the directed edge $\langle w_0, v_0\rangle$ in a face node;
2. **while** $(\langle v_0, x\rangle \neq \langle v_0, w_0\rangle)$ **do**
2.1. let $\langle v_0, y\rangle$ be the directed edge following the directed edge $\langle x, v_0\rangle$ in a face node;
2.2. **if** ($\langle v_0, y\rangle$ is marked) **then** stop('not manifold')
2.3. **else** mark $\langle v_0, y\rangle$; $x = y$;
3. mark $\langle v_0, w_0\rangle$.

Algorithm F-Validity (D**)** {* D is a DLFL structure *}

1. by scanning the face list, edge list, and vertex list, check that each edge correctly
corresponds to two directed edges;
2. **for** (each vertex v in D) **do**
if (v is not an isolated vertex without edge)
then in v's vertex list, find an directed edge $\langle v, w\rangle$ from v;
VertexTrace$(v, \langle v, w\rangle)$;
3. **if** (there are still unmarked directed edges) **then** return('not manifold').

Figure 5. Topological validity testing on DLFL structure

A *doubly linked face list* (DLFL) for a mesh structure G consists of a *face list* F, an *edge list* E, and a *vertex list* V. Each *face node* in the face list F is a cyclically and doubly linked list of directed edges, lined up head to tail to form a boundary walk of the face. Each *vertex node* in the vertex list V is a list of pointers to the corresponding vertices in the face nodes in F. Each *edge node* in the edge list E consists of two bi-directed pointers to the directed edges induced from the edge in the face nodes in F. (See appendix foran example of DLFL data structure.)

Both vertex list and edge list provide efficient access to directed edges in the face list. The testing algorithm for Validity Rules on the DLFL structure is given in Figure 5.

Theorem 4.3 *If a DLFL structure D can pass the test of algorithm* **F-Validity***, which takes time $O(m)$, then D represents a valid 2-manifold.*

PROOF. Validity Rule 1 is tested by step 1 of the main algorithm **F-Validity**.

To verify Validity Rule 2, note that two consecutive directed edges $\langle v, w\rangle$ and $\langle w, x\rangle$ in a face boundary walk specify precisely that $\langle w, x\rangle$ is the directed edge following the directed edge $\langle w, v\rangle$ in the rotation at w. Thus, the subroutine **VertexTrace**$(v_0, \langle v_0, w_0\rangle)$ starts from a directed edge $\langle v_0, w_0\rangle$ and constructs a list of consecutive directed edges from v_0. If the face list does not give a valid rotation

at a vertex v_0, then there are only two possibilities: (1) a directed edge at v_0 follows two different directed edges at v_0. This will be detected by step 2.2 in the subroutine **VertexTrace** when the directed edge is encountered the second time; (2) the directed edges at v_0 form more than one disjoint cyclically ordered lists. This will be detected by step 3 of the main algorithm **F-Validity** since the call to the subroutine **VertexTrace**$(v_0, \langle v_0, w_0\rangle)$ can trace only one such a list.

In conclusion, if the DLFL structure passes the test of **F-Validity**, then it must represent a valid graph rotation system. For the complexity of the algorithm, note that the algorithm **F-Validity** marks each directed edge exactly once. Since the edge node for an edge $[v, w]$ has two bi-directed pointers to the directed edges $\langle v, w\rangle$ and $\langle w, v\rangle$ in the face list, from an directed edge $\langle v, w\rangle$ in the face list, we can find the directed edge $\langle w, v\rangle$ in constant time. Therefore, the algorithm **F-Validity** runs in time $O(m)$. □

Before we close this section, we give a general remark on the implementation of our validity testing algorithms. Our algorithms check the topological validity of mesh structures based on directed edges. Therefore, for data structures that explicitly reserve a physical record for each directed edge in their representations, our algorithms can be directly applied. Note that most modeling data structures, including the half-edge structure, the quad-edge structure, and the DLFL structure, do this. For these data structures, we can simply attach for each directed edge an additional bit to its record, which will be used to mark if the directed edge has been traced in our algorithms. On the other hand, if a data structure does not reserve such records for directed edges, such as the winged-edge structure, then the implementation of our algorithms must be more careful. Weiler [26] has thoroughly studied the problem of traversing a mesh on the winged-edge structure. In particular, since the winged-edge structure does not explicitly give directed edges in its representation, when a mesh under winged-edge structure is traversed, an additional piece of information must be kept to indicate which directed edge induced by an edge is being referred. The situation can become even worse when multiple edges and self-loops are allowed in a mesh [26]. Therefore, for winged-edge structures, we suggest that our algorithms be implemented on the modified winged-edge structures in which directed edges are explicitly recorded, as proposed by Weiler [26].

Also, since graph rotation systems only describe cellular mesh structures, for systems that also allow non-cellular meshes, our algorithms must be used with care.

5 Manifold preserving operations

Operations on mesh modeling are important, in particular for the modeling systems with a reliable and powerful user-interface. A systematic study on the operator set on mesh modeling was given by Mäntylä [19], and was also considered by other researchers (e.g., [7, 14]). For 2-manifold modeling, three issues in modeling operator sets have turned out to be the most important: the completeness, the soundness, and the efficiency. We say that a set S of operators is *complete* if any 2-manifold mesh structure can be generated by a sequence of the operators from S, and that the set S of operators is *sound* if applying any operator in S on a 2-manifold results in a valid 2-manifold structure.

Theorem 3.1 provides a solid foundation not only for validity testing of 2-manifold structures, but also for powerful mesh modeling operators. In the previous studies on operators on 2-manifold structures (e.g., [14] and [19]), it is always needed to verify carefully that a proposed set of operators is sound and complete so that all and only 2-manifold structures can be created using the operators. On the other hand, by Theorem 3.1, the problem of soundness and completeness can be resolved based on the representation of graph rotation systems: as long as we can verify that a set of operators can create all and only valid graph rotation systems, the soundness and completeness of the set are guaranteed. Note that validity of graph rotation systems is a simple graph property, which is in general much easier and more intuitive than that of mesh topological integrity.

5.1 A complete and sound operator set

We study a set of modeling operators [5] that seems simpler, more intuitive and user-friendly than the previously proposed ones. We will discuss the soundness and completeness of the operator set, consider its efficient implementations on a variety of modeling data structures, and compare it in terms of various aspects with the existing approaches.

The first operator is *edge insertion*, briefly INSERTEDGE. Let $\rho(G)$ be a graph rotation system and let u and v be two vertices in G. By inserting a new edge $[u, v]$ between two face corners $cor(\langle u, x\rangle, \langle u, x'\rangle)$ and $cor(\langle v, y\rangle, \langle v, y'\rangle)$, we mean inserting the directed edge $\langle u, v\rangle$ in the rotation at u between the directed edges $\langle u, x\rangle$ and $\langle u, x'\rangle$, and inserting the directed edge $\langle v, u\rangle$ in the rotation at v between the directed edges $\langle v, y\rangle$ and $\langle v, y'\rangle$. Note that if the two face corners belong to the same face, then the new inserted edge "splits" the face into two faces On the other hand, if the two face corners belong to different faces, then the new inserted edge "merges" the two faces into a single face. The inverse of the INSERTEDGE operator is the *edge deletion* operator, briefly DELETEEDGE. Let $\rho(G)$ be a rotation system and let $e = [u, v]$ be an edge in G, by deleting the edge e, we mean deleting the directed edge $\langle u, v\rangle$ from the rotation at the vertex u and deleting the directed edge $\langle v, u\rangle$ from the rotation at the vertex v.

The other primary operator is *vertex creation*, briefly CREATEVERTEX, which creates a new vertex without edge in the rotation system. The inverse of CREATEVERTEX is DELETEVERTEX, which removes a vertex without edge from the rotation system.

Remark. Compared to the Euler operators studied in [19], CREATEVERTEX and DELETEVERTEX are similar to the Euler operators MVFS and KVFS, and INSERTEDGE and DELETEEDGE are similar to the Euler operators MEF and KEF, and MEKR and KEMR (in some sense, also MFKRH and KFMRH). The other Euler operators, such as MEV and KEV, have no correspondence in our proposed set of operators (see [19] for more detailed explanations for Euler operators).

Theorem 5.1 *The operator set consisting of the operators* INSERTEDGE/DELETEEDGE *and* CREATEVERTEX/DELETEVERTEX *is complete and sound.*

PROOF. The soundness of the operator set is obvious: applying each of the four operators on a graph rotation system that represents a valid 2-manifold structure results in a valid graph rotation system, which by Theorem 3.1 is a valid 2-manifold structure.

The completeness can also be verified easily. Any cellular mesh structure induces a unique rotation system $\rho(G)$. By a sequence of DELETEEDGE followed by a sequence of DELETEVERTEX, we can obtain an empty rotation system. Reverse this sequence of operators, and replace in the sequence each DELETEVERTEX by a CREATEVERTEX and each DELETEEDGE by an INSERTEDGE. The resulting sequence starting from an empty rotation system should reconstruct the rotation system $\rho(G)$. Therefore, any cellular mesh structure can be obtained by a sequence of operators in the set. □

Compared to the Euler operators proposed in [7, 19], our set consists of far fewer operators. Compared with the operator set proposed by Guibas and Stolfi [14], our set consists of the same number of operators but seems more intuitive (we suggest that readers compare our INSERTEDGE/DELETEEDGE operators with the SPLICE operator proposed in [14]). Moreover, our set of operators seems more uniform, from either the viewpoint of modeling systems or the viewpoint of users. A modeling system using our operators only needs to deal with its internal representation on graph rotation systems, without any concern of the topological integrity of the underlying mesh, while a user using our operators only needs to identify face corners on the mesh structure, without any need of understanding its internal implementation. Most previously proposed operator sets seem to have difficulties to completely avoid the interlacing between the system level and the user level, and between the internal representations and the topological integrity.

5.2 Implementation on vertex-based data structures

The implementation of the operators INSERTEDGE, DELETEEDGE, CREATEVERTEX, and DELETEVERTEX on an adjacency list structure is fairly simple. For INSERTEDGE to insert an edge $[u, w]$ between face corners $cor(\langle u, x\rangle, \langle u, x'\rangle)$ and $cor(\langle w, y\rangle, \langle w, y'\rangle)$, we only need to insert the item w between the items x and x' in the linked list for the vertex u, and insert the item u between the items y and y' in the linked list for the vertex w. DELETEEDGE that deletes an edge $[u, w]$ from the adjacency list structure is implemented by deleting the item w in the linked list for u and deleting the item u in the linked list for w. Finally, a CREATEVERTEX operator corresponds to adding a new vertex with an empty list in the adjacency list, and DELETEVERTEX corresponds to removing a vertex with an empty list from the adjacency list.

The CREATEVERTEX and DELETEVERTEX implemented in an adjacency list take constant time. However, the operators INSERTEDGE and DELETEEDGE implemented this way may have to search through the linked lists for two vertices, which can be time-consuming in particular when the vertices have large valences. The time complexity of the operators INSERTEDGE and DELETEEDGE can be improved if we use a balanced tree instead of a linked list for the directed edges from each vertex, which supports item insertion and item deletion in the tree in time logarithmic to the size of the tree [8]. Moreover, in this implementation, a supplementary edge list will be needed to support efficient searching of directed edges in the structure. We leave the detailed implementation of this alternative data structure to the interested readers.

5.3 Implementation on edge-based data structures

Again we discuss the implementation on the winged-edge structure, and describe briefly its extension to other structures such as half-edge structure and quad-edge structure.

In order to implement the CREATEVERTEX and DELETEVERTEX operators, we need to introduce a supplementary vertex list in the winged-edge structure (actually, such a supplementary vertex list is always needed in an edge-based data structure when isolated vertices without edges are allowed). Each item v in the vertex list, called a *vertex node*, has a pointer to an (arbitrary) edge node whose one end is v. A vertex node for an isolated vertex without edges has an empty pointer. Now, CREATEVERTEX simply

Subroutine InsertEdge($[u,w], cor(\langle u,x\rangle, \langle u,x'\rangle)$, $cor(\langle w,y\rangle, \langle w,y'\rangle)$**)**
{* Inserting edge $e = [u,w]$ to face corners $cor(\langle u,x\rangle, \langle u,x'\rangle)$ and $cor(\langle w,y\rangle, \langle w,y'\rangle)$. *}

1. create a new edge node $A[e]$ for $e = [u,w]$ with $A[e].$*vstart* $= u$ and $A[e].$*vend* $= w$;
2. find the edge nodes $e_1 = [u,x]$, $e_2 = [u,x']$, $e_3 = [w,y]$, and $e_4 = [w,y']$;
   {* without loss of generality, assume $A[e_1].$*vstart* $= A[e_2].$*vstart* $= u$ and
   $A[e_3].$*vstart* $= A[e_4].$*vstart* $= w$. Other cases can be handled similarly. *}
3. $A[e_1].$*nccw* $= e$; $A[e_2].$*pcw* $= e$; $A[e].$*nccw* $= e_2$; $A[e].$*pcw* $= e_1$;
4. $A[e_3].$*ncw* $= e$; $A[e_4].$*pccw* $= e$; $A[e].$*ncw* $= e_4$; $A[e].$*pccw* $= e_3$;
5. call **FaceTrace**$(\langle u,w\rangle)$ to update the components *fcw* and *fccw*;
6. **if** (the directed edge $\langle w,u\rangle$ is not traced in step 5)
   then call **FaceTrace**$(\langle w,u\rangle)$ to update the components *fcw* and *fccw*.

Subroutine DeleteEdge(e**)**

1. find the edge node $A[e]$ for the edge $e = [u,w]$, where $A[e].$*vstart* $= u$ and $A[e].$*vend* $= w$;
2. find the edge nodes $e_1 = A[e].$*pcw*, $e_2 = A[e].$*nccw*, $e_3 = A[e].$*pccw*, and $e_4 = A[e].$*ncw*;
   {* without loss of generality, assume $A[e_1].$*vstart* $= A[e_2].$*vstart* $= u$, and
   $A[e_3].$*vstart* $= A[e_4].$*vstart* $= w$. Other cases are handled similarly. *}
3. $A[e_1].$*nccw* $= e_2$; $A[e_2].$*pcw* $= e_1$; $A[e_3].$*ncw* $= e_4$; $A[e_4].$*pccw* $= e_3$;
4. remove the edge node $A[e]$;
5. call **FaceTrace**$(\langle u,x'\rangle)$ to undate the face components *fcw* and *fccw*;
6. **if** (the directed edge $\langle w,y'\rangle$ is not traced in step 5)
   then call **FaceTrace**$(\langle w,y'\rangle)$ to update the components *fcw* and *fccw*.

Figure 6. INSERTEDGE **and** DELETEEDGE **on winged-edge structure**

adds a new vertex node with an empty pointer to the structure, and DELETEVERTEX removes a vertex node with an empty pointer from the structure.

To implement the operators INSERTEDGE and DELETEEDGE, consider in the winged-edge structure an edge node $A[e]$ with components *name*, *vstart*, *vend*, *ncw*, *pcw*, *nccw*, *pccw*, *fcw*, and *fccw* (see Figure 9). The components *nccw* and *pcw* give the "next" and "previous" directed edges to the directed edge $\langle$*vstart*, *vend*$\rangle$, respectively, while the components *ncw* and *pccw* give the "next" and "previous" directed edges to the directed edge $\langle$*vend*, *vstart*$\rangle$, respectively. Finally, the components *fcw* and *fccw* give the "right face" and "left face" while traveling from vertex *vstart* to vertex *vend* along the edge.

Implementations of the operators INSERTEDGE and DELETEEDGE on a winged-edge structure are given in Figure 6. By the definitions of the components *ncw*, *pcw*, *nccw*, and *pccw* (see Figure 9), it is easy to verify that the operator INSERTEDGE inserts the directed edge $\langle u,w\rangle$ to the face corner $c_1 = cor(\langle u,x\rangle, \langle u,x'\rangle)$ and inserts the directed edge $\langle w,u\rangle$ to the face corner $c_2 = cor(\langle w,y\rangle, \langle w,y'\rangle)$, while the operator DELETEEDGE deletes the edge e. The components *ncw*, *pcw*, *pcw*, and *pccw* of the related edges are properly updated. The only thing that needs further explanation is the updating of the components *fcw* and *fccw*. Consider INSERTEDGE. If the two face corners c_1 and c_2 belong to different faces, then inserting the edge $e = [u,w]$ between c_1 and c_2 replaces the two faces in the mesh by a single new face whose boundary contains both directed edges $\langle u,w\rangle$ and $\langle w,u\rangle$ of the inserted edge e. Therefore, in this case, the subroutine **FaceTrace**$(\langle u,w\rangle)$ in step 5 will trace the boundary of this new face and properly update the face components of the related edge nodes. In particular, the subroutine **FaceTrace** in step 6 will not be executed. On the other hand, if the face corners c_1 and c_2 belong to the same face in the mesh, then inserting the edge e between c_1 and c_2 replaces the face by two new faces whose boundaries contain the two directed edges $\langle u,w\rangle$ and $\langle w,u\rangle$ of the inserted edge e, respectively. Thus, the subroutine **FaceTrace** in step 5 traces one new face and updates the face components for the related edge nodes, while the subroutine **FaceTrace** in step 6 traces the other new face and updates the face components for the related edges nodes.

For the operator DELETEEDGE on an edge e, if the two directed edges of e are on boundaries of two different faces, then deleting e will "merge" the two faces into a single one. Thus, the subroutine **FaceTrace** in step 5 will trace this new face and update the face components of the related edge nodes. On the other hand, if the two directed edges of e are on the boundary of the same face, then deleting e will "split" this face into two faces Now the subroutine **FaceTrace** in step 5 and step 6 will trace the two new faces and update the face components of the related edge nodes.

Now we consider the complexity of the algorithms INSERTEDGE and DELETEEDGE. For updating the components *ncw*, *nccw*, *pcw*, and *pccw*, since only very few edges (two or three) are involved, this can be done in constant time. However, each algorithm contains one or two executions of the subroutine **FaceTrace**, whose time complexity is proportional to the size of the involved faces. When the face size is small (such as on a triangulated mesh), this is fine. However, for faces of large size, this will be time consuming.

In section 4.2 we remarked that the components *name*, *vstart*, *vend*, *ncw*, and *nccw* in the winged-edge structure are sufficient for representing a 2-manifold mesh structure. In particular, a winged-edge structure without the components *fcw* and *fccw* is sufficient for describing a 2-manifold mesh structure. Moreover, on a winged-edge structure without the components *fcw* and *fccw*, step 5 and step 6 in the algorithms INSERTEDGE and DELETEEDGE are not needed. We summarize this discussion in the following theorem.

Theorem 5.2 *On a general winged-edge structure, the running time of the algorithms* INSERTEDGE *and* DELETEEDGE *is* $O(s)$*, where* s *is the size of the faces involved,*

while on a winged-edge structure without the components fcw and fcw, the running time of the algorithms INSERTEDGE *and* DELETEEDGE *is* $O(1)$.

The discussion of the implementations of the operators INSERTEDGE and DELETEEDGE on the winged-edge structure can be easily extended to other edge-based data structures, such as the half-edge structure and the quad-edge structure, since these data structures have components that are similar to the components *ncw*, *nccw*, *pcw*, *pccw*, *fcw*, and *fccw* in the winged-edge structure.

Finally, we would like to compare our operators INSERTEDGE and DELETEEDGE with the SPLICE operator proposed by Guibas and Stolfi [14]. Roughly speaking, the SPLICE operator is equivalent to first inserting an edge between two face corners then "contracting" the edge, while the inverse of the SPLICE operator is equivalent to first splitting a vertex then deleting the edge between the two new vertices. Therefore, for the SPLICE operator and our INSERTEDGE operator, and for the inverse operator of the SPLICE operator and our DELETEEDGE operator, roughly the same number of edges get involved in the updating of the components *ncw*, *nccw*, *pcw*, *pccw*, *fcw*, and *fccw*. However, the SPLICE operator and its inverse operator also need to change the end-vertices of the involved edges, which can be time consuming in particular when the involved vertices have high valence. Moreover, our operators are purely topological, while SPLICE needs to combine two vertices, which necessarily requires changing the geometric positions of the vertices, and in consequence, complicates the modeling process [12].

5.4 Implementation on face-based data structures

As we pointed out, the INSERTEDGE and DELETEEDGE operators implemented on the adjacency list data structure can become time consuming when the involved vertices have high valence, while those implemented on the winged-edge structure or its variations can become time consuming when the involved faces have large size. Removing the components *fcw* and *fccw* from the winged-edge structure can speedup the running time, but may make it inconvenient since then directed edges will not be directly associated with their corresponding faces. In the following, we consider implementations of our operators on the DLFL structure, and show that the computation inefficiency can be removed.

To implement CREATEVERTEX(v) in DLFL, we create a face node in the face list with a degenerated boundary of a single vertex v and a vertex node in the vertex list with a pointer to the new face. Similarly, to implement DELETEVERTEX(v), we remove the corresponding face node and vertex node in DLFL.

The algorithms for INSERTEDGE and DELETEEDGE are given in Figure 7. The correctness of the algorithm INSERTEDGE implemented on the DLFL structure follows from the proof of Theorem 3.1 (see [9]). The algorithm DELETEEDGE reverses the process of INSERTEDGE so its correctness also follows. For the complexity of the algorithms, since each edge node in the edge list has two pointers to its two directed edges in the face list, from each edge, its two directed edges in the face list can be accessed in constant time. If we implement the list of directed edges in each face node as a balanced tree [1], then testing whether two directed edges belong to the same face, splitting one directed edge list into two directed edge lists, and merging two directed edge lists into a single directed edge list can all be done in time $O(\log s)$, where s is the size of the involved faces (for more detailed description and verification, see [8]). Therefore, under this implementation for face nodes, the running time of the algorithms INSERTEDGE and DELETEEDGE is bounded by $O(\log s)$. This concludes our discussion.

Theorem 5.3 *Under the above implementation on DLFL, the operators* CREATEVERTEX *and* DELETEVERTEX *take time* $O(1)$*, and the operators* INSERTEDGE *and* DELETEEDGE *take time* $O(\log s)$*, where* s *is the size of the involved faces.*

6 Conclusions and final remarks

In this paper, we have investigated some fundamental issues, including representations, data structures, and operators, for 2-manifold mesh modeling. We extended the theory of graph rotation systems and showed that the extended theory provides a solid foundation for 2-manifold mesh modeling. Based on this fundamental result, we examined existing data structures and proposed a new data structure for 2-manifold mesh modeling. These data structures are either vertex-based, edge-based, or face-based. We developed simple and efficient algorithms for testing the topological validity of meshes on these data structures. We also proposed a new set of operators on 2-manifold mesh modeling and proved that all and only 2-manifold mesh structures can be constructed by sequences of these operators. Efficient implementations of our operators on vertex-based, edge-based, and face-based data structures were presented. Advantages of our operator set over the operator sets previously proposed in the literature were indicated. To close the paper, we offer some further remarks.

Our fundamental theorem was proved for meshes in which multiple edges and self-loops are allowed. For simplicity, most our algorithms were presented only for meshes without multiple edges and self-loops. It is very straightforward to extend our algorithms to mesh structures with multiple edges and self-loops. In fact, in a mesh with no multiple edges and self-loops, a directed edge is given as

Subroutine InsertEdge($[u,w], cor(\hat{e}'_1,\hat{e}'_2), cor(\hat{e}''_1,\hat{e}''_2)$)

1. make two directed edges $\hat{e} = \langle u, w\rangle$ and $r(\hat{e}) = \langle w, u\rangle$ of the edge $e = [u, w]$;
2. **if** (the face corners $cor(\hat{e}'_1,\hat{e}'_2)$ and $cor(\hat{e}''_1,\hat{e}''_2)$ belong to the same face
$f = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t))$
then delete the face node f from the face list;
add two new face nodes to the face list:
$f' = (r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1))$ and $f'' = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}'_1))$;
add an edge node $[u, w]$ in the edge list and set its two pointers properly;
else {* the face corners $cor(\hat{e}'_1,\hat{e}'_2)$ and $cor(\hat{e}''_1,\hat{e}''_2)$ belong to the two different faces
$f_1 = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s)$ and $f_2 = (r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t)$ *}
delete the face nodes f_1 and f_2 from the face list;
add a new face node to the face list:
$f = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}''_1), r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}'_1))$;
add an edge node $[u, w]$ in the edge list and set its two pointers properly.

Subroutine DeleteEdge(e)

1. find the two directed edges $\hat{e}$ and $r(\hat{e})$ of the edge e in the face list;
2. **if** (the two directed edges $\hat{e}$ and $r(\hat{e})$ belong to two different faces
$f' = (r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1))$ and $f'' = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}'_1))$;
then delete the face nodes f' and f'' from the face list;
add a new face node to the face list:
$f = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t)$;
delete the edge node e from the edge list;
else {* the two directed edges $\hat{e}$ and $r(\hat{e})$ belong to the same face
$f = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}''_1), r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}'_1))$.
*};
delete the face node f from the face list;
add two new face nodes to the face list:
$f_1 = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s)$ and $f_2 = (r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t)$;
delete the edge node e in the edge list.

Figure 7. INSERTEDGE **and** DELETEEDGE **on DLFL structure**

$\langle u, w\rangle$, where u and w are the two end-vertices of the edge. When multiple edges and self-loops are allowed, then we only need to give each edge two distinguishable directed edges. With this modification, all algorithms can be translated to work on meshes with multiple edges and self-loops.

In some applications, it is also desired that the "dual mesh" is given in a data structure for a mesh structure [14]. Briefly speaking, each face in the original mesh corresponds to a vertex in the dual mesh and each vertex in the original mesh corresponds to a face in the dual mesh. Two vertices in the dual mesh are adjacent if and only if the two corresponding faces in the original mesh share a common boundary edge. Most importantly, traveling the boundary of a face in the original mesh corresponds to traveling the directed edges in the rotation at the corresponding vertex in the dual mesh (and vice versa). Our fundamental Theorem 3.1 points out that in fact, for most existing data structures for mesh modeling, the dual mesh structure is already given, explicitly and implicitly. For example, in the vertex-based adjacency list data structure for meshes, the rotation given in the linked list for each vertex actually gives the boundary traveling of the corresponding face in the dual mesh, while the rotation at each vertex of the dual mesh can be recovered by calling the algorithm **FaceTrace** on the corresponding face in the original mesh. In the edge-based winged-edge structure, we only need to re-interpret the components *vstart*, *vend*, *fcw* and *fccw*, and exchange the vertex components and the face components. Note that the components *ncw*, *nccw*, *pcw*, and *pccw* provide sufficient information not only for the involved vertices but also for the involved faces. Finally, in the face-based DLFL structure, each face node in the face list, which is a cyclically linked list of boundary directed edges of the face, gives exactly the rotation at the corresponding vertex in the dual mesh. Therefore, the face-based face list in the DLFL structure for the original mesh in fact gives a vertex-based adjacency list for the dual mesh. By Theorem 4.1, the dual mesh structure can be constructed based on this list.

So far we have been focused on modeling orientable manifolds. Our research can be extended to modeling orientable manifolds with boundaries and to modeling non-orientable manifolds. First, as described in [19] (chapter 3), modeling 2-manifolds can be extended to handle 2-manifolds with boundaries. For modeling non-orientable manifolds, we point out that the *edge algebra* model and its company quad-edge data structure [14] can handle both orientable and non-orientable surfaces. For our models, there is a simple extension based on graph rotation systems to represent non-orientable surfaces. Roughly speaking, if we also label each edge in a graph rotation system by a "twisted/untwisted" mark, then the extended rotation system can be used to represent non-orientable manifolds. Following the same manner, we can prove that this extension always represents valid (orientable and non-orientable) 2-manifold structures, that the existing data structures for orientable 2-manifold mesh modeling can be augmented to deal with non-orientable 2-manifold mesh modeling, and that slight modifications on the operators in our operator set form a complete and sound operator set for orientable and non-orientable mesh modeling. See [10] for some related theoretical results.

Based on the theoretical results studied in the current paper, we have developed a prototype topological mesh modeler [4, 5, 6]. Our prototype modeler has become phenomenal success among our students [28], for the following reasons:

1. *Programmer's Point of View:* using the proposed operator set, it is very easy for programmers to extend the capabilities of our system by writing simple codes for new high level user operators. Our recently developed

works include automatic filling of missing polygons [24], curved handle creation [23], rind modeling [6], wire modeling [17], and various new but yet unpublished shape modeling tools.

2. *User's Point of View:* It is extremely easy to use our system. Although topology change is considered a challenging mathematical concept, the users of our system can easily learn it and create complicated meshes. Figures 11, 12 and , 13 in appendix show examples 2-manifold meshes created by our students as their weekly projects in a regular graduate level shape modeling course, using our software (see [28] for more examples).

References

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] E. AKLEMAN AND J. CHEN, Guaranteeing the 2-manifold property for meshes with doubly linked face list, *International J. of Shape Modeling 5*, (1999), pp. 149-177.

[3] E. AKLEMAN, J. CHEN, F. ERYOLDAS, AND V. SRINIVASAN, A new corner cutting scheme with tension and handle-face reconstruction, *International J. of Shape Modeling 7*, (2001), pp. 111-128

[4] E. AKLEMAN, J. CHEN, AND V. SRINIVASAN, A new paradigm for changing topology during subdivision modeling, *Proc. 8th Pacific Conf. on Computer Graphics and Applications*, (PG'2000), (2000), pp. 192-201.

[5] E. AKLEMAN, J. CHEN, AND V. SRINIVASAN, A prototype system for robust, interactive and user-friendly modeling of orientable 2-manifold meshes, *Proc. International Conf. on Shape Modeling and Applications*, (SM'02), (2002), pp. 43-50.

[6] E. AKLEMAN, V. SRINIVASAN, AND J. CHEN, Interactive rind modeling, *Proc. International Conf. on Shape Modeling and Applications*, (SM'03), (2003), pp. 23-32.

[7] B. J. BAUMGART, Winged-edge polyhedron representation, *Technical Report CS-320*, (1972), Stanford University.

[8] J. CHEN, Algorithmic graph embeddings, *Theoretical Computer Science 181*, (1997), pp. 247-266.

[9] J. CHEN AND E. AKLEMAN, Topologically robust mesh modeling: concepts, data structures and operations, *Technical Report*, Department of Computer Science, Texas A&M University, College Station, TX (2003), also available at `www-viz.tamu.edu/faculty/ergun/topology`.

[10] J. CHEN, J. L. GROSS, AND R. G. RIEPER, Overlap matrices and total imbedding distributions, *Discrete Mathematics 128*, (1994), pp. 73-94.

[11] J. EDMONDS, A combinatorial representation for polyhedral surfaces, *Notices American Mathematics Society 7*, (1960), p. 646.

[12] M. GARLAND, Quadric-based polygonal surface simplification, *Ph.D. Dissertation*, Computer Science Department, Carnegie Mellon Univ., CMU-CS-99-105, May 1999.

[13] J. L. GROSS AND T. W. TUCKER, *Topological Graph Theory*, Wiley Interscience, New York, 1987.

[14] L. GUIBAS AND J. STOLFI, Primitives for the manipulation of general subdivision and computation of Voronoi diagrams, *ACM Transaction on Graphics 4*, (1985), pp. 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), pp. 101-165.

[17] E. MANDAL AND E. AKLEMAN, Wire modeling, *Visual Proc. of SIGGRAPH'03* (Art and Design Sketch), to appear, San Diego, CA (2003).

[18] M. MÄNTYLÄ, Boolean operations on 2-manifolds through vertex neighborhood classification, *ACM Transaction on Graphics 5*, (1986), pp. 1-29.

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

[20] MAYA, Maya Manual, Alias Wavefront, Toronto, Canada, 1999.

[21] T. M. MURALI AND T. A. FUNKHOUSER, Consistent solid and boundary representations from arbitrary polygonal data, *Proc. 1997 Symposium on Interactive 3D Graphics*, (1997), pp. 155-162.

[22] SPATIAL TECHNOLOGY INC., ACIS 3D Toolkit 5.0, (1999).

[23] V. SRINIVASAN, E. AKLEMAN, AND J. CHEN, Interactive construction of multi-segment curved handles, *Proc. 10th Pacific Conf. on Computer Graphics and Applications*. (2002), pp. 429-430.

[24] V. SRINIVASAN, E. AKLEMAN, AND J. KEYSER, Automatically filling cracks and missing polygons, *Manuscript*, (2003).

[25] UNIGRAPHICS SOLUTIONS INC., http://www.parasolid.com.

[26] K. WEILER, Topological structures for geometric modeling, *Ph.D. Dissertation*, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[27] D. ZORIN, ed., *Subdivision for Modeling and Animation*, ACM SIGGRAPH'2000 Course Notes no.23, 2000.

[28] Computer Aided Sculpting class webpages:
www-viz.tamu.edu/courses/viza657/01fall/homework/hw04.html
www-viz.tamu.edu/courses/viza657/02fall/homework/hw03.html
www-viz.tamu.edu/courses/viza657/02fall/homework/hw05.html
www-viz.tamu.edu/courses/viza657/03fall/homework/viza657.2.html

A Appendix: Supplemental Materials

These material will not to be included with our final submission

B Commercial Modeling System Example

An example of a commercial modeling system that allow non-manifold structures

We take a widely used modeling system, MAYA [20] as an example. Consider the shapes in Figure 8. From the 2-

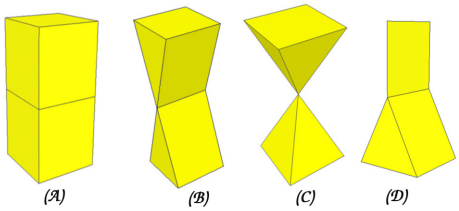


Figure 8. Examples of non-manifolds that can be created in Maya.

manifold shape in Figure 8(A), non-manifold structures in Figure 8(B) and Figure 8(C) can be created in MAYA by using the MERGEVERTICES operator, and the non-manifold structure in Figure 8(D) can be created in MAYA by using EXTRUDEEDGE operator. Moreover, MAYA allows users to reverse normal of a face, which can also result in non-manifold structures.

C Winged Edge Structure

Formally, the main component of a winged-edge data structure is a list of *edge nodes*. Each edge node consists of nine components: (*name, vstart, vend, fcw, fccw, ncw, pcw, nccw, pccw*), where *name* is the name of the edge; *vstart* and *vend* are the starting and ending end-vertices of the edge (which thus give a "direction" to the edge); *fcw* and *fccw* indicate the right face and the left face of the edge when traversing along the edge direction; *ncw* and *pccw* give the next edges in the faces *fcw* and *fccw* when traversing along the edge direction; and *pcw* and *nccw* give the next edges in the faces *fcw* and *fccw* when traversing along the reversed direction of the edge. See Figure 9 for illustration.

C.1 DLFL Data Structure

Figures 10 gives a partial part of a DLFL data structure for a tetrahedron.

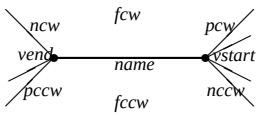


Figure 9. A edge node in the winged-edge structure

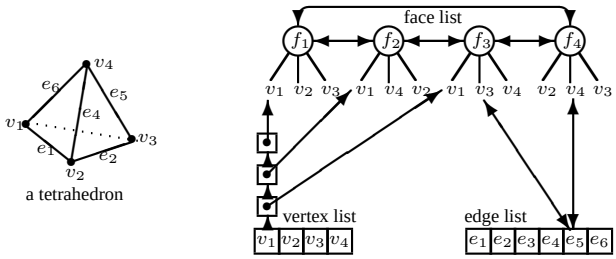


Figure 10. The DLFL data structure for a tetrahedron

D Examples of Student Work

Figures 11, 12 and , 13 show examples 2-manifold meshes created by our students as their weekly projects in a regular graduate level shape modeling course, using our software (see [28] for more examples).

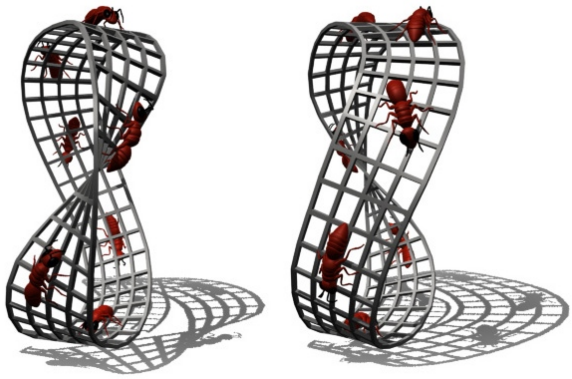


Figure 11. Escher's Möbius band by Avneet Kaur

E Proofs for Theorem 3.1 and Theorem 4.2

We first show how to reconstruct faces on a given graph rotation system. Let $\rho(G)$ be a rotation system of the graph

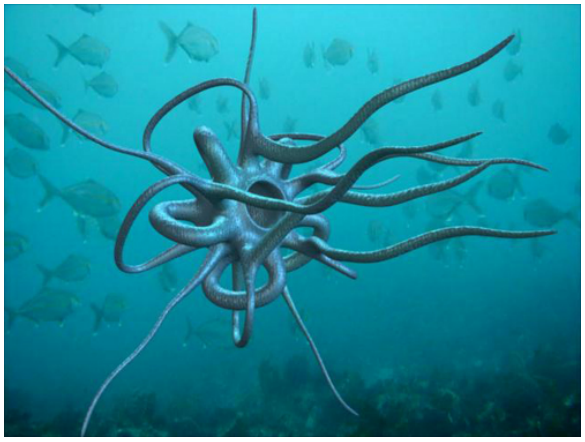


Figure 12. Sea scene by Hyemee Choi.

Figure 13. Wired Models by Esan Mandal. Orginal Bogart Caricature by Han Lei.

Subroutine FaceTrace($\langle u_0, w_0\rangle$) {* $\langle u_0, w_0\rangle$ is a directed edge *}

1. trace $\langle u_0, w_0\rangle$;
2. let $\langle u, w\rangle$ be the directed edge next to $\langle w_0, u_0\rangle$ in the rotation at w_0, where $u = w_0$;
3. **while** ($\langle u, w\rangle \neq \langle u_0, w_0\rangle$) **do**
   let $\langle w, x\rangle$ be the directed edge next to $\langle w, u\rangle$ in the rotation at w;
   trace $\langle u, w\rangle$; $\langle u, w\rangle = \langle w, x\rangle$.

Algorithm Facing ($\rho(G)$) {* $\rho(G)$ is a graph rotation system *}

1. **for** each isolated vertex v without incident edges in G **do**
   construct a face with a degenerated boundary v;
2. **while** there is a directed edge $\langle u_0, w_0\rangle$ not traced yet **do** FaceTrace($\langle u_0, w_0\rangle$).

Figure 14. Face reconstruction based on graph rotation systems

G. The algorithm presented in Figure 3 is re-given in Figure 14, where "tracing" a directed edge is either to mark or to store the directed edge, in its given direction.

Lemma A.1 *The algorithm* **Facing** *traces each directed edge in $\rho(G)$ exactly once.*

PROOF. First of all, each directed edge is traced at least once. This is because if any directed edge $\hat{e}$ is not traced, then the main algorithm will call the subroutine **FaceTrace** on $\hat{e}$, which will trace the edge. Thus, it suffices to show that no directed edge is traced more than once.

Consider the subroutine **FaceTrace($\hat{e}_0$)**, where $\hat{e}_0 = \langle u_0, w_0\rangle$. Let $\hat{e} = \langle u, w\rangle$ be a directed edge and let $\hat{e}' = \langle u, x\rangle$ be the directed edge right before $\hat{e}$ in the rotation at the vertex u. If $\hat{e}$ is traced by **FaceTrace($\hat{e}_0$)** and $\hat{e} \neq \hat{e}_0$, then it is clear by the algorithm **FaceTrace($\hat{e}_0$)** that the directed edge $r(\hat{e}') = \langle x, u\rangle$ must be also traced by **FaceTrace($\hat{e}_0$)**, right before the tracing of the directed edge $\hat{e}$. On the other hand, if $\hat{e} = \hat{e}_0$, then the directed edge $\hat{e} = \hat{e}_0$ is traced at beginning of **FaceTrace($\hat{e}_0$)** and **FaceTrace($\hat{e}_0$)** stops when the directed edge $\hat{e} = \hat{e}_0$ is encountered the second time. Thus, when $\hat{e} = \hat{e}_0$, the last edge traced by **FaceTrace($\hat{e}_0$)** must be the directed edge $r(\hat{e}') = \langle x, u\rangle$.

Now assume that some of the directed edges in $\rho(G)$ are traced more than once by the algorithm **Facing**. Suppose that $\hat{e}$ is the directed edge that receives the earliest second time trace. First note that the second time trace on the directed edge $\hat{e}$ cannot be the first edge trace of a subroutine execution **FaceTrace($\hat{e}_0$)**: in this case $\hat{e}$ would be $\hat{e}_0$ and the second time trace on $\hat{e}$ is the first edge trace of **FaceTrace($\hat{e}_0$)**, but when **FaceTrace($\hat{e}_0$)** is called, the directed edge $\hat{e}_0$ should have not been traced at all.

Suppose that the directed edge $\hat{e}$ receives the first time trace in **FaceTrace($\hat{e}_1$)**, and receives the second time trace in **FaceTrace($\hat{e}_2$)**. Let $\hat{e} = \langle u, w\rangle$, and let $\hat{e}' = \langle u, x\rangle$ be

the directed edge right before $\hat{e}$ in the rotation at the vertex u. We consider two cases.

If $\hat{e}_1 \neq \hat{e}_2$, then by the above remarks, the directed edge $r(\hat{e}') = \langle x, u\rangle$ also receives a trace in **FaceTrace($\hat{e}_1$)**. Now since the second time trace on $\hat{e}$ cannot be the first edge trace in **FaceTrace($\hat{e}_2$)**, the second time trace on $\hat{e}$ in **FaceTrace($\hat{e}_2$)** must follow a trace on the directed edge $r(\hat{e}')$. Thus, the directed edge $r(\hat{e}')$ would receive the second time trace earlier than $\hat{e}$.

If $\hat{e}_1 = \hat{e}_2$, then the directed edge $\hat{e}$ cannot be $\hat{e}_1$ since the subroutine **FaceTrace($\hat{e}_1$)** stops immediately when the edge $\hat{e}_1$ is encountered the second time. Thus, the first time trace and the second time trace on the directed edge $\hat{e}$ must both follow traces on the directed edge $r(\hat{e}') = \langle x, u\rangle$. This, again, shows that the directed edge $r(\hat{e}')$ would receive the second time trace before the second time trace on the directed edge $\hat{e}$.

Thus, in any case, we would derive that the directed edge $r(\hat{e}')$ would receive the second time trace before the second time trace on the directed edge $\hat{e}$. This contradicts the assumption that the directed edge $\hat{e}$ receives the earliest second time trace, thus showing that no edge should receive a second time trace. The lemma is proved. □

By Lemma A.1, each subroutine execution **FaceTrac($\hat{e}_0$)** produces a sequence of directed edges that line up head to tail and form a polygonal closed walk. One thing we should show is that this construction is robust, i.e., the order in which the main algorithm **Facing** picks untraced edges should not affect the output of the algorithm.

Lemma A.2 *On a directed edge $\hat{e}_1$, the subroutine* **FaceTrace($\hat{e}_1$)** *produces a sequence S of directed edges that form a polygonal closed walk. Moreover, for any directed edge $\hat{e}_2$ in the sequence S, the subroutine* **FaceTrace($\hat{e}_2$)** *on $\hat{e}_2$ will produce the same polygonal closed walk.*

PROOF. By the algorithm, if a directed edge $\langle u, w\rangle$ is traced in **FaceTrace**, then the next edge in the closed walk is uniquely determined, which must be the directed edge right after $\langle w, u\rangle$ in the rotation at the vertex w. Moreover, the first edge $\langle u, w\rangle$ traced by **FaceTrace** must be the directed edge right after the last directed edge traced by **FaceTrace** in the rotation at vertex u. Therefore, regardless of the starting directed edge in the closed walk, the same polygonal closed walk will be produced. □

Therefore, for a given graph rotation system $\rho(G)$, the algorithm **Facing** will produce a unique set of point-spheres, each for an isolated vertex in G, and a unique set of polygonal closed walks. We will regard each of the polygonal closed walks as the boundary of a face. According to Lemma A.1 and Lemma A.2, the set of faces constructed this way is uniquely determined. Thus, the graph rotation system $\rho(G)$ plus the algorithm **Facing** specifies a unique mesh structure by giving the vertex set (that is the vertex set of the graph G), the edge set (that is the edge set of the graph G), and the face set (that is given by the algorithm **Facing**). We show below that the mesh structure constructed by this process is cellular and gives a unique and valid 2-manifold structure. For this purpose, we need to first discuss two operations on graph rotation systems.

Let v be a vertex in the graph G. For any two consecutive directed edges $\hat{e}_1 = \langle v, u\rangle$ and $\hat{e}_2 = \langle v, w\rangle$ in the rotation at v, the two directed edges $r(\hat{e}_1)$ and $\hat{e}_2$ are consecutive on the boundary of a face f constructed by the algorithm **Facing**. we say that the directed edges $\hat{e}_1$ and $\hat{e}_2$ form a *face corner* (of the face f) at v, written either as $cor(\hat{e}_1, \hat{e}_2)$ or $cor(u, v, w)$. In case v is an isolated vertex without incident edges, we define that v has a unique face corner $cor(v)$. Note that the two directed edges may belong to the same edge (e.g., both of them are from the same self-loop). In an extreme case, the two directed edges $\hat{e}_1$ and $\hat{e}_2$ can even be the same directed edge (i.e., when the vertex v is of valence 1).

Let $\rho(G)$ be a rotation system of a graph G, and let $e = [u, w]$ be an edge in G. By deleting the edge from $\rho(G)$, we obtain naturally a rotation system for the graph $G - e$. More formally, let $\hat{e}' = \langle u, w\rangle$ and $\hat{e}'' = \langle w, u\rangle$ be the two directed edges induced from the edge e in $\rho(G)$, and suppose that the rotations in $\rho(G)$ at u and w are (recall that the directed edges are cyclically ordered in the rotation at each vertex):

$$u: \ \hat{e}'\hat{e}'_1\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \ \hat{e}''\hat{e}''_1\hat{e}''_2\cdots\hat{e}''_m$$

where $\hat{e}'_i$ and $\hat{e}''_j$ are directed edges. Then the resulting rotation system after deleting e from $\rho(G)$ has the identical rotation at each vertex $x \neq u, w$, and at u and w, the rotations are

$$u: \ \hat{e}'_1\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \ \hat{e}''_1\hat{e}''_2\cdots\hat{e}''_m$$

Conversely, suppose in the rotation system $\rho(G)$, the directed edges $\hat{e}'_1$ and $\hat{e}'_2$ form a face corner at a vertex u, and the directed edges $\hat{e}''_1$ and $\hat{e}''_2$ form a face corner at a vertex w, then we can *insert* a new edge $e = [u, w]$ between the face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$ in the rotation system $\rho(G)$ to make a rotation system for the graph $G + e$. More formally, suppose that in the rotation system $\rho(G)$, the rotations at the vertices u and w are

$$u: \ \hat{e}'_1\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \ \hat{e}''_1\hat{e}''_2\cdots\hat{e}''_m$$

respectively. After inserting the edge e between the face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$, we obtain a rotation

system $\rho(G+e)$ for the graph $G+e$ that has the identical rotation at each vertex $x \neq u, w$, and at the vertices u and w, the rotations are

$$u: \quad \hat{e}'_1\hat{e}'\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \quad \hat{e}''_1\hat{e}''\hat{e}''_2\cdots\hat{e}''_m$$

where $\hat{e}' = \langle u, w\rangle$ and $\hat{e}'' = \langle w, u\rangle$ are the two directed edges of the edge e (or more precisely, we actually inserted the directed edge $\hat{e}'$ to the face corner $cor(\hat{e}'_1, \hat{e}'_2)$ and the directed edge $\hat{e}''$ to the face corner $cor(\hat{e}''_1, \hat{e}''_2)$).

Lemma A.3 *A graph rotation system $\rho(G)$ plus the face set constructed by the algorithm* **Facing** *on $\rho(G)$ gives a cellular mesh structure on a unique and valid 2-manifold.*

PROOF. We prove the lemma by induction on the number of edges in the graph G. Suppose the graph G has n vertices.

If the graph G has no edges, then all vertices of G are isolated. Thus, the algorithm **Facing** uniquely produces n point-spheres, which form a valid 2-manifold on which $\rho(G)$ gives a valid cellular mesh structure.

Now assume that the graph G has $m > 0$ edges. Let $e = [u, w]$ be an edge in G (u and w may belong to the same vertex if e is a self-loop). Suppose that in the rotation system $\rho(G)$, the rotations at the vertices u and w are:

$$u: \quad \hat{e}'_1\hat{e}\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \quad \hat{e}''_1 r(\hat{e})\hat{e}''_2\cdots\hat{e}''_m$$

where $\hat{e} = \langle u, w\rangle$ and $r(\hat{e}) = \langle w, u\rangle$ are the two directed edges of the edge e. By deleting the edge $e = [u, w]$ in the rotation system $\rho(G)$, we obtain a rotation system $\rho(G-e)$ for the graph $G-e$ of $m-1$ edges. In particular, the rotations at the vertices u and w in the rotation system $\rho(G-e)$ become

$$u: \quad \hat{e}'_1\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \quad \hat{e}''_1\hat{e}''_2\cdots\hat{e}''_m$$

(it is possible that one or both of the rotations become an empty rotation now).

By the inductive hypothesis, the rotation system $\rho(G-e)$ plus the algorithm **Facing** uniquely gives a valid 2-manifold structure S_-. Let F_- be the set of faces in the mesh $\rho(G-e)$. Let $f_1 \in F_-$ be the face containing the face corner $cor(\hat{e}'_1, \hat{e}'_2)$ and let $f_2 \in F_-$ be the face containing the face corner $cor(\hat{e}''_1, \hat{e}''_2)$. By inserting the edge $e = [u, w]$ between the face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$ in $\rho(G-e)$, we get back the rotation system $\rho(G)$. We show that $\rho(G)$ also gives a valid 2-manifold structure.

Apply the algorithm **Facing** on the rotation system $\rho(G)$. Note that since the only difference between $\rho(G-e)$ and $\rho(G)$ is at the face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$, the algorithm **Facing** on $\rho(G)$ will produce all faces in $F_- - \{f_1, f_2\}$. Since the set of faces produced by **Facing** is independent of the order of the edges, we can assume

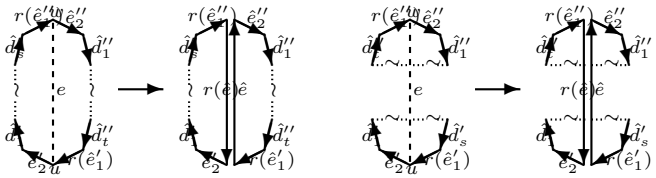


Figure 15. Left: inserting an edge between two corners of the same face; Right: inserting an edge to face corners of two different faces

that the algorithm **Facing** on $\rho(G)$ first constructs all faces in $F_- - \{f_1, f_2\}$. Note that after the construction of these faces, the only directed edges that have not been traced are those on the boundaries of the faces f_1 and f_2, and the two directed edges $\hat{e}$ and $r(\hat{e})$ induced by the edge $e = [u, w]$. There are two possible cases.

Case 1. The two face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$ belong to the same face in $\rho(G-e)$, that is $f_1 = f_2$. Let the boundary of the face $f_1 = f_2$ be

$$f_1 = f_2 = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t)$$

where $\hat{d}'_i$ and $\hat{d}''_j$ are directed edges. Since the rotations at any vertex $x \neq u, w$ are identical in the rotation systems $\rho(G-e)$ and $\rho(G)$, and the rotations at u and w in $\rho(G)$ are

$$u: \quad \hat{e}'_1\hat{e}\hat{e}'_2\cdots\hat{e}'_n \qquad \text{and} \qquad w: \quad \hat{e}''_1 r(\hat{e})\hat{e}''_2\cdots\hat{e}''_m,$$

the procedure **FaceTrace(**$r(\hat{e})$**)** on the rotation system $\rho(G)$ will produce the face

$$f' = (r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}''_1))$$

and the procedure **FaceTrace(**$\hat{e}$**)** on $\rho(G)$ produces the face

$$f'' = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}'_1))$$

(see the left figure in Figure 15 for illustration). Therefore, the difference of the two meshes $\rho(G)$ and $\rho(G-e)$ is that the face $f_1 = f_2$ in $\rho(G-e)$ is replaced by the faces f' and f'' in $\rho(G)$. By looking at the face structures, we can easily see that the face $f_1 = f_2$ can be obtained by "pasting" the faces f' and f'' along the edge e. Therefore, adding the face $f_1 = f_2$ to the mesh and adding the two faces f' and f'' to the mesh should give the same structure. Since the mesh constructed from $\rho(G-e)$ represents the valid 2-manifold S_-, we conclude that the mesh constructed from $\rho(G)$ should also represent the same 2-manifold S_-. In fact, it is not difficult to see that the mesh $\rho(G)$ can be obtained from the mesh $\rho(G-e)$ by adding a crossing edge e on the face $f_1 = f_2$. Thus, in this case, the mesh $\rho(G)$ and $\rho(G-$

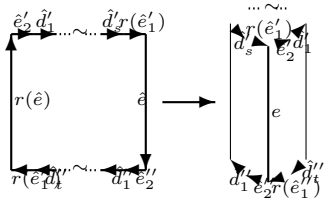


Figure 16. Pasting the two directed edges of an edge to get a cylinder

Subroutine VertexTrace(v_0, e_0) {* the vertex v_0 is an end of the edge e_0 *}

1. **if** ($A[e_0]$.*vstart* $= v_0$) **then** $e = A[e_0]$.*nccw* **else** $e = A[e_0]$.*ncw*;
2. **while** ($e \neq e_0$) **do**
2.1. suppose $e = [v_0, w]$;
2.2. **if** (the directed edge $\langle v_0, w\rangle$ has been marked) **then** Stop('not manifold');
2.3. mark the directed edge $\langle v_0, w\rangle$;
2.4. **if** ($A[e]$.*vstart* $= v_0$) **then** $e = A[e]$.*nccw* **else** $e = A[e]$.*ncw*;
3. suppose $e_0 = [v_0, w_0]$; mark the directed edge $\langle v_0, w_0\rangle$.

Algorithm E-Validity ($A[1..m]$) {* $A[1..m]$ is a winged-edge structure *}

for each edge node $A[e]$ **do**
1. $u = A[e]$.*vstart*; $w = A[e]$.*vend*;
2. **if** (u is not marked) { **then** mark u; VertexTrace(u, e); }
else if (the directed edge $\langle u, w\rangle$ has not been marked) **then** Stop('not manifold');
3. **if** (w is not marked) **then** { mark w; VertexTrace(w, e); }
else if (the directed edge $\langle w, u\rangle$ has not been marked) **then** Stop('not manifold').

Figure 17. Topological validity testing on winged-edge structure

e) represent the same manifold. In other word, inserting the edge e to $\rho(G-e)$ does not change the manifold topology.

Case 2. The two face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$ belong to two different faces f_1 and f_2, $f_1 \neq f_2$, in $\rho(G-e)$. Let the boundaries of the two faces f_1 and f_2 be

$$f_1 = (r(\hat{e}'_1), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s) \qquad \text{and} \qquad f_2 = (r(\hat{e}''_1), \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t)$$

Again since the rotation system $\rho(G)$ is obtained from the rotation system $\rho(G-e)$ by inserting the edge e between the two face corners $cor(\hat{e}'_1, \hat{e}'_2)$ and $cor(\hat{e}''_1, \hat{e}''_2)$, when the procedure **FaceTrace($\hat{e}'$)** is called, we will get the face:

$$f = (\hat{e}, \hat{e}''_2, \hat{d}''_1, \ldots, \hat{d}''_t, r(\hat{e}''_1), r(\hat{e}), \hat{e}'_2, \hat{d}'_1, \ldots, \hat{d}'_s, r(\hat{e}'_1))$$

(see the right figure in Figure 15 for illustration.) Note that the boundary of the face f has contained all directed edges on the boundaries of the faces f_1 and f_2, plus the two directed edges of the edge e. Therefore, the difference between the mesh $\rho(G)$ and the mesh $\rho(G-e)$ is that the faces f_1 and f_2 in $\rho(G-e)$ are replaced by the face f in $\rho(G)$. Now if we past along the edge e in the face f, we will get a cylinder Y with two open ends. (See Figure 16 in which we have re-drawn the face f then pasted along the two directed edges $\hat{e}$ and $r(\hat{e})$.) Observe that if we traverse the two open ends of the cylinder Y, we get exactly the boundary traversings of the faces f_1 and f_2 in $\rho(G-e)$. Therefore, the structure represented by the mesh $\rho(G)$ can be obtained from the 2-manifold S_- represented by $\rho(G-e)$ as follows: first cut the two faces f_1 and f_2 off along their boundaries in S_-, then on the resulting structure past the boundaries of the two open ends of the cylinder Y along the boundaries of the two faces f_1 and f_2. It is well-known that such a topological surgery on a 2-manifold results in a 2-manifold [13]. Thus, in this case, the mesh $\rho(G)$ also represents a valid manifold structure.

In terms of the topological structure, case 2 can be further divided into two subcases: (1) the faces f_1 and f_2 belong to the same surface in the manifold – in this subcase, inserting the edge e to $\rho(G-e)$ (i.e., running a tube between the two faces f_1 and f_2) adds a handle to the surface, thus the number of surfaces in the manifold is unchanged while the manifold genus is increased by 1; (2) the faces f_1 and f_2 belong to different surfaces in the manifold – in this subcase, inserting the edge e connects the two surfaces by a tube, thus the number of surfaces in the manifold is decreased by 1 while the manifold genus is unchanged. In any case, inserting the edge e in case 2 changes the manifold topology.

Summarizing the above discussion, we have proved that in any case, the mesh $\rho(G)$ represents a valid 2-manifold structure. This completes the proof of the lemma. □

As we described before, each mesh structure on a 2-manifold also naturally induces a graph rotation system. Combining this fact with Lemma A.3, we have completed the proof for the following fundamental theorem.

Theorem 3.1 *A mesh structure defines a valid 2-manifold if and only if it uniquely specifies a graph rotation system $\rho(G)$, whose face set is constructed by the algorithm* **Facing**$(\rho(G))$ *in Figure 14.*

Now we prove Theorem 4.2. The algorithm **E-Validity** presented in Figure 4 is re-given in Figure 17.

Theorem 4.2 *If a winged-edge structure $A[1..m]$ can pass the test of the algorithm* **E-Validity** *in Figure 17, then $A[1..m]$ represents a valid 2-manifold structure. Moreover, the test algorithm runs in time $O(m)$.*

PROOF. An edge node $A[e]$ uniquely induces two directed

edges $\langle A[e].\textit{vstart}, A[e].\textit{vend}\rangle$ and $\langle A[e].\textit{vend}, A[e].\textit{vstart}\rangle$. Thus, Validity Rule 1 is satisfied.

For each edge node $A[e]$, the component $A[e].\textit{nccw}$ uniquely induces the directed edge next to the directed edge $\langle A[e].\textit{vstart}, A[e].\textit{vend}\rangle$ in the rotation at $A[e].\textit{vstart}$, and the component $A[e].\textit{ncw}$ uniquely induces the directed edge next to the directed edge $\langle A[e].\textit{vend}, A[e].\textit{vstart}\rangle$ in the rotation at $A[e].\textit{vend}$. Therefore, the winged-edge structure $A[1..m]$ defines a "NEXT" operation on directed edges in $A[1..m]$ such that if $\text{NEXT}(\langle v, w\rangle) = \langle v, x\rangle$, then $\langle v, x\rangle$ is the directed edge next to $\langle v, w\rangle$ in the rotation at the vertex v. To verify Validity Rule 2, we only need to check whether this NEXT operation arranges the directed edges at each vertex into a cyclic ordering. Since the operation NEXT gives a unique next directed edge for each directed edge, and since for a directed edge $\hat{e}$, $\text{NEXT}(\hat{e})$ and $\hat{e}$ are from the same vertex, there are only two possibilities that the NEXT operation does not arrange the directed edges at a vertex v into a cyclic ordering: (1) a directed edge $\hat{e}$ is next to two different directed edges $\hat{e}_1$ and $\hat{e}_2$, i.e., $\text{NEXT}(\hat{e}_1) = \text{NEXT}(\hat{e}_2) = \hat{e}$; or (2) the NEXT operation arranges the directed edges at a vertex into more than one disjoint cyclically ordered lists. Both these conditions can be identified effectively by the algorithm **E-Validity**, as follows. If for two different directed edges $\hat{e}_1$ and $\hat{e}_2$, we have $\text{NEXT}(\hat{e}_1) = \text{NEXT}(\hat{e}_2) = \hat{e}$, then the directed edge $\hat{e}$ (or some other directed edge) will be encountered again after its first marking. Thus, step 2.2 in the subroutine **VertexTrace** will detect this and correctly report an error. If the NEXT operation arranges the directed edges at a vertex v into more than one disjoint cyclically ordered lists, then after the first call on the subroutine **VertexTrace**(v, e), the vertex v is marked while some directed edges from v are not marked. This situation will be detected by either step 2 or step 3 of the main algorithm **E-Validity**, which will report correctly the error.

On the other hand, if the components *ncw* and *nccw* correctly describe a valid graph rotation system, then each execution of the subroutine **VertexTrace**(v, e) will mark the vertex v as well as all directed edges from v, which can in fact also produce the cyclic order of the directed edges from v (this part is not presented in the algorithm). Thus, no error will be reported by either step 2.2 in the subroutine **VertexTrace** or by the step 2 or step 3 in the main algorithm **E-Validity**.

Thus, if the winged-edge structure $A[1..m]$ passes the test of **E-Validity**, then the components *vstart*, *vend*, *ncw* and *nccw* of the edge nodes in the winged-edge structure describe a valid graph rotation system. By the Validity Rules, the theorem is proved.

Now we consider the complexity of the test for the validity of a winged-edge structure. Suppose that for a given edge e, the edge node $A[e]$ can be accessed in constant time (this condition can be easily met if the edge nodes are arranged in an array $A[1..m]$ where the edge names are given by the array indices), then the algorithm **E-Validity** marks each vertex and each directed edge in constant time. Recording whether a vertex or a directed edge has been marked can be easily done by using a supplementary vertex array and a supplementary edge array). In conclusion, the running time of the algorithm **E-Validity** is bounded by $O(m + n) = O(m)$, where m is the number of edges and n is the number of vertices in the winged-edge structure. □