Nonmanifold Topology Based on Coupling Entities

Yasushi Yamaguchi and Fumihiko Kimura
University of Tokyo

A nonmanifold topology model can surmount the limitations of traditional solid models. We propose a data structure and two classes of operations, focusing on neighborhood relationships and boundary information.

Geometric modeling is an important technique for building CAD/CAM and other engineering application systems. Many kinds of geometric models exist, such as wireframe, surface, and solid models. Some consider a solid model to be a complete modeling framework, because every real object has its own volume. However, it has some problems. For instance, a regularized Boolean operation may yield a nonmanifold result. Furthermore, it turns out that a solid model is not always relevant for some sorts of applications. A manifold solid model can only represent one closed volume minus its internal structure. Also, it cannot represent a mixed geometry of curves, surfaces, and volumes, which can occur in the early stages of design. A nonmanifold model, on the other hand, can provide a suitable framework for representing form features.

Topology models aim at handling incident relationship information. We can categorize recent studies on nonmanifold topology by how they represent incidence-ordering information. The three kinds of cyclic ordering in 3D topology are loop, disk, and radial (Figure 1).[1,2] The loop cycle (Figure 1a) corresponds to the face-edge cycle in solid models. The radial cycle (Figure 1b) is a cycle of faces connected to a certain edge. The disk cycle (Figure 1c) resembles the vertex-edge cycle in solid models. However, a nonmanifold vertex may have several disk cycles, whereas a solid-model vertex has only one vertex-edge cycle. Radial edge representation (RER)[3] can represent a radial cycle as well as a loop cycle. Vertex-based representation (VBR)[2] can represent a disk cycle, a radial cycle, and a loop cycle all at once.

Several works start from Voronoi diagram representations. Dobkin proposed facet-edge data structure (FEDS)[4] for manipulating 3D subdivisions. Brisson[1] and Lienhardt[5] subsequently proposed cell tuple structure (CTS) and n-dimensional generalized map (n-G-map), respectively. Although their mathematical background makes these representations rigorous, their power to represent product shapes is very limited. For instance, they cannot handle faces and regions having several boundaries or dangling edges and faces. Furthermore, the units of the representations—a facet edge in FEDS, a cell tuple in CTS, or a dart in n-G-map—are much more primitive than those of RER and VBR. In other words, RER and VBR can provide more compact representation than FEDS, CTS, or n-G-map.

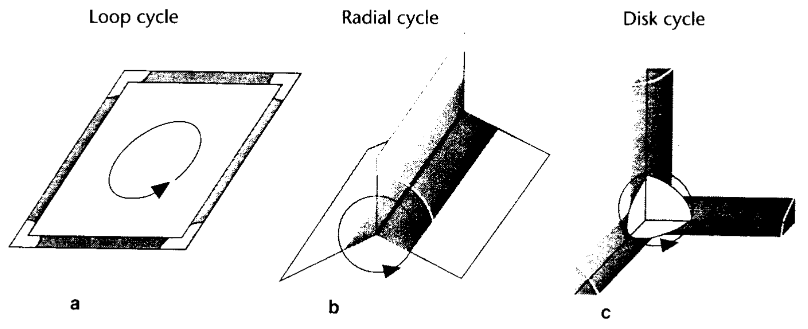
Our study

The domain of our study is cell decomposition in a 3D Euclidean space, E^3. The study concentrated on representing and manipulating ordering information with compact representation. We examined these issues that affect a nonmanifold model in E^3.

Data entities. In solid modeling, a data entity can be a vertex, edge, face, shell, or loop. To represent incidence-ordering information, nonmanifold topology calls for some additional types of data entities. For instance, Weiler introduced faceuse, loopuse, edgeuse, and vertexuse in RER. VBR also required additional types of data entities. RER and VBR can actually represent two and three types of cycles, respectively. However, nobody has determined what additional types of data entities are necessary to represent incidence relationships.

Pointers. A set of pointers among data entities describes incidence-ordering information. Two-manifold topology uses vertex-edge and face-edge cycles. Weiler pointed out that four types of edge based data structures could represent those cycles: Face-Edge, Vertex-Edge, Winged-Edge, and Modified-Winged-Edge.[6] These differ from each other in their pointer sets. Nonmanifold topology requires much more complicated pointer sets. RER represents a loop cycle with previous/next edgeuse pointers and a radial cycle with

0272-17-16/95/$4.00 © 1995 IEEE



1 The three types of cycles in nonmanifold topology: (a) loop, (b) radial, and (c) disk.

radial/mate pointers. VBR uses tricyclic pointers to represent three kinds of cycles. It seems that a number of pointer sets can represent incidence-ordering information.

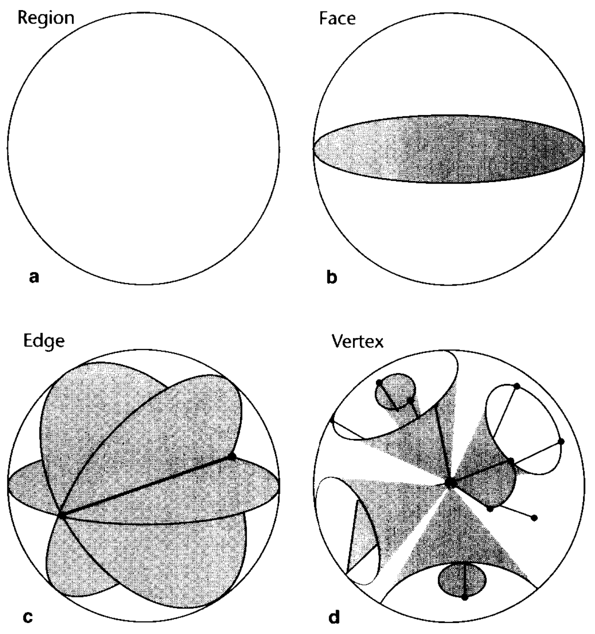
Singularities. A nonmanifold model must be able to represent mixed geometry composed of points, curves, surfaces, and volumes. Thus, a model may contain several singular cases. For instance, consider an isolated vertex on a face. The vertex forms a loop, and the loop has a pointer to the vertex. However, in general a loop has a pointer to an edge, because a loop is usually composed of a sequence of edges. The data structure must be able to represent this kind of singular case by clearly defining a regular case and distinguishing singular cases from it.

Euler operations. Euler operations in a solid model are well known, as they guarantee its topological consistency.[7-9] Masuda et al. proposed Euler operations for nonmanifold topology that provide an interpretation of the Euler-Poincaré formula for a cell complex.[10] An Euler operation corresponds to a transition vector that keeps satisfying the formula, such as mEV (make_Edge_Vertex). In general, interpretations for nonmanifold topology are more complicated than those for a manifold solid, and complicated interpretations cause more variations of Euler operations. Furthermore, many variations of Euler operations correspond to one transition vector. Thus, nonmanifold topology has many kinds of Euler operations, and implementation is not an easy task.

Operations for additional entities. The interpretations of the Euler-Poincaré formula substitute variables only for numbers of conventional data entities such as vertices, edges, and faces. The Euler-Poincaré formula cannot constrain additional data entities such as vertexuse or edgeuse; it also does not present any criteria for manipulating those additional entities. Thus, when and how the additional data entities should be created or deleted is left unaddressed.

Data structure

This section covers our data structure for nonmanifold topology. The first thing we must do is introduce data entities standing for point sets and their relations. Then we discuss pointer sets for representing incidence-ordering.



2 These diagrams describe neighborhoods of (a) a point in a region, (b) a point on a face, (c) an edge point, and (d) a vertex.

Data entity

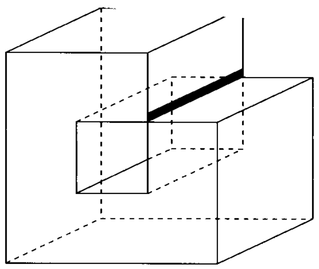
The objective of our nonmanifold model is to represent point sets and their incidence relation in E^3. Thus, we start by introducing primitive entities standing for point sets. A *primitive entity* corresponds to a cell without boundary, an open set of points bounded by lower dimensional cells. We can classify four types of primitive entities according to their underlying dimensions: A vertex corresponds to a 0-cell, an edge corresponds to a 1-cell, a face corresponds to a 2-cell, and a region corresponds to a 3-cell.

A conventional representation of a manifold solid model is based on boundary representation, in which a solid is bounded by shells consisting of faces, a face is bounded by loops consisting of edges, and an edge is bounded by one or two vertices. This study, on the other hand, also uses neighborhood classification. We define the neighborhood of a point as an intersection with a ball of infinitesimal radius having its center on the point. Figure 2 illustrates neighborhoods of points in primitive entities.

The nonmanifold model must be able to describe this kind of neighboring information. Specifically, the same

Table 1. Determining the type of coupling entities that join primitive entity pairs.

	Vertex	Edge	Face	Region
Vertex	***	end	fan (ring)	corner (ball)
Edge	end	***	blade	wedge (tube)
Face	fan (ring)	blade	***	side
Region	corner (ball)	wedge (tube)	side	***



3 A nonmanifold edge, shown by a bold line, is incident to four faces and four regional portions, though the number of actual regions are only two.

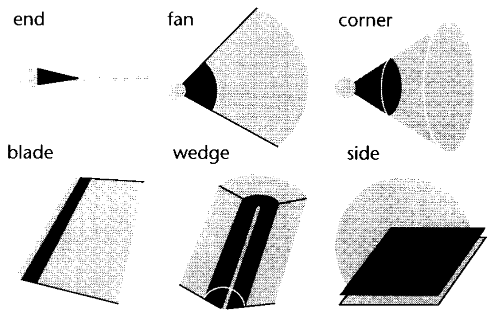
pair of primitive entities can intersect each other several times. For example, only two regions are incident to the edge shown by a bold line in Figure 3, while four regional portions are incident to the edge. We introduced the *coupling entity* to stand for a portion of incidence.[11] The six types of coupling entities are end, fan, corner, blade, wedge, and side, composed according to the types of the incident pair (see Table 1). A neighborhood of a point of each primitive entity will be represented in terms of coupling entities. Figure 4 illustrates the coupling entities.

Because boundary information is very important for most applications, the model must be able to represent boundaries. Some entities might have several disjoint boundaries. A *bounding entity* stands for an individual boundary that is also composed of coupling entities. A region can be bounded by several shells, a face can be bounded by several loops, and a corner can be bounded by several disks (Figure 5).

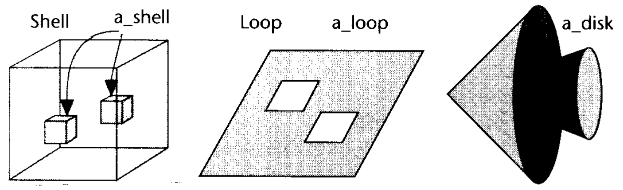
Our model does not require an explicit representation of a fan, blade, or wedge. The new coupling entity we named *feather* stands simultaneously for a side of a fan, a side of a blade, and an end of a wedge. A pair of feathers represents a fan, blade, or wedge. A feather looks like an edgeuse in RER[3] or a cusp in VBR,[2] as Figure 6 shows. However, a feather is neither a part of an edge nor a part of a vertex. It is an intermediate entity for representing a fan, blade, and wedge.

Pointer set

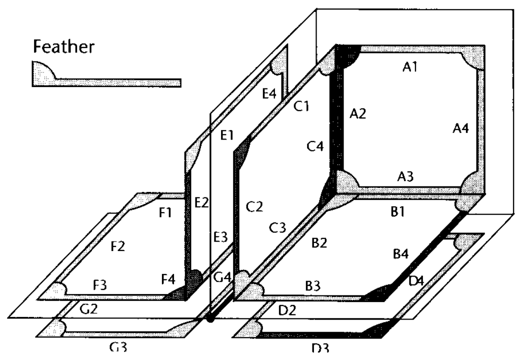
A feather needs several pointers to represent the three kinds of cycles. Our original idea was to represent cycles in terms of coupling entities. Thus, we wanted feather pointers to represent both a fan/blade/wedge and the next entities in the cycle. These requirements imply two categories of pointers, *mate pointers* and *cyclic pointers*. A mate pointer represents a fan/blade/wedge pair by pointing to that entity's mate. Mate pointers fall into one of three categories according to the coupling pair they represent: fan mate (FM), blade mate (BM), and wedge mate (WM).



4 The six types of coupling entities join pairs of primitive entities into a neighborhood.



5 Bounding entities represent boundaries consisting of coupling entities.



Fan	
B4 & D3	:D3 = FM(B4), B4 = FM(D3)
Blade	
C2 & E2	:E2 = BM(C2), C2 = BM(E2)
Wedge	
A2 & C4	:C4 = WM(A2), A2 = WM(C4)
Loop cycle	
A1>>A2>>A3>>A4	:A2 = CCL(A1), A3 = CCL(A2)
A4>>A3>>A2>>A1	:A1 = CL(A2), A2 = CL(A3)
Radial cycle	
D2>>C3>>F4	:C3 = CCR(D2), F4 = CCR(C3)
F4>>C3>>D2	:D2 = CR(C3), C3 = CR(F4)
Disk cycle	
A3>>C4>>B2	:C4 = CCD(A3), B2 = CCD(C4)
B2>>C4>>A3	:A3 = CD(C4), C4 = CD(B2)

6 Our method combines feathers into pairs to take the place of separate definitions for fans, blades, and wedges.

A cyclic pointer points to the next entity on a cycle. Thus six types of cyclic pointers represent certain cycles and cycle orientations: counterclockwise loop (CCL), clockwise loop (CL), counterclockwise radial (CCR), clockwise radial (CR), counterclockwise disk (CCD), and clockwise disk (CD). Figure 6 shows some examples of pointer relations. FM(x) denotes the fan mate feather of a feather x. CCL(x) denotes the next counterclockwise feather on a loop of a feather x. BM(x), CL(x), and so on are defined in a similar manner.

Since a mate pointer points to entities' mates, the relations $x = \mathrm{FM}(\mathrm{FM}(x)) = \mathrm{BM}(\mathrm{BM}(x)) = \mathrm{WM}(\mathrm{WM}(x))$ are obvious. A pair of cyclic pointers, CCX and CX, are inversely related as follows:

$$x = \mathrm{CCL}(\mathrm{CL}(x)) = \mathrm{CL}(\mathrm{CCL}(x)) \quad (1)$$
$$x = \mathrm{CCR}(\mathrm{CR}(x)) = \mathrm{CR}(\mathrm{CCR}(x)) \quad (2)$$
$$x = \mathrm{CCD}(\mathrm{CD}(x)) = \mathrm{CD}(\mathrm{CCD}(x)) \quad (3)$$

Furthermore, the following dependencies occur among the pointers:

$$\mathrm{CCD(x)} = \mathrm{CCR(FM(x))} \quad (4)$$
$$\mathrm{CCR(x)} = \mathrm{WM(BM(x))} \quad (5)$$
$$\mathrm{CCL(x)} = \mathrm{FM(BM(x))} \quad (6)$$

The above six equations indicate dependencies among the nine types of pointers. If we select three of them to be independent, we can deduce the remaining six relations from the three pointers. Thus we need at least three pointers to represent cycles.

We chose FM, BM, and WM as our three basic pointers for the following reasons.

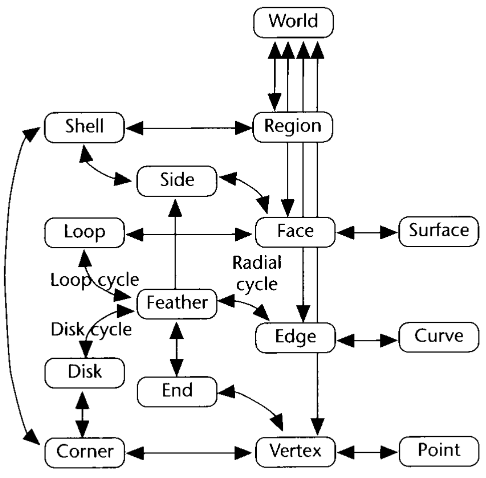
- Sufficiency. The three pointers are independent. The rest can be deduced from them as follows:

$$\begin{aligned} \mathrm{CCL}(x) &= \mathrm{FM}(\mathrm{BM}(x)) \\ \mathrm{CL}(x) &= \mathrm{BM}(\mathrm{FM}(x)) \\ \mathrm{CCR}(x) &= \mathrm{WM}(\mathrm{BM}(x)) \\ \mathrm{CR}(x) &= \mathrm{BM}(\mathrm{WM}(x)) \\ \mathrm{CCD}(x) &= \mathrm{WM}(\mathrm{BM}(\mathrm{FM}(x))) \\ \mathrm{CD}(x) &= \mathrm{FM}(\mathrm{BM}(\mathrm{WM}(x))) \end{aligned}$$

- Efficiency. The total number of steps needed to access the nine relations is the smallest among all the pointer sets.
- Convenience. For writing codes, it is convenient that the inverse relations are symmetrically accessed. Furthermore, the three pointers explicitly represent coupling entities—fans, blades, and wedges—which is very useful for implementing topological operations.

Let us consider other nonmanifold representations. According to our notation of feather pointers, an RER[3] edgeuse has four pointers, CL, CCL, BM, and WM. A VBR[2] cusp has three pointers, CCL, CR, and CCD. Both pointer sets are sufficient. RER's pointer set is convenient but not efficient—it is redundant. On the other hand, the VBR's pointer set is efficient, but not convenient for programming.

Figure 7 diagrams the proposed data structure. The data structure consists of four types of primitive entities, three types of bounding entities, and four types of coupling entities. Using feathers to represent the three types of cycles instead of fans, blades, and wedges eliminates the need for those entities. The three basic pointers of a feather—FM, BM, and WM—can trace the cycles.



7 Our data structure replaces three of the original six coupling entities—fans, blades, and wedges—with feather pairs.

Operations

This section explains two classes of topological operations, called *Euler operations* and *neighborhood operations*. These operations guarantee to satisfy certain constraints on nonmanifold topology. They form a complete set of operations so that they can manipulate all types of data entities in a consistent way.

Euler-Poincaré formula

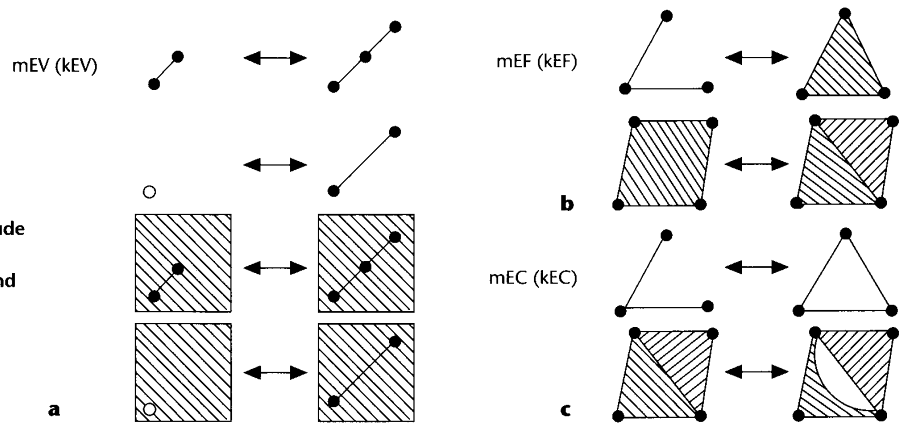
The Euler-Poincaré formula for cell complexes, on which Euler operations are based, is

$$\sum_{\gamma=0}^{n} (-1)^{\gamma} \alpha_{\gamma} = \sum_{\gamma=0}^{n} (-1)^{\gamma} p_{\gamma} \quad (7)$$

Here, n denotes the dimension, α_γ denotes the number of γ-cells, and p_γ denotes the γ-Betti number. Since the domain of the topological representation is cell decomposition in E^3, we want to examine the 2D case of Equation 7, which contains all types of primitive entities: vertex, edge, face, and region. The variables can be interpreted as follows: α_0 = number of vertices, α_1 = number of edges, α_2 = number of faces, p_0 = number of connected components, p_1 = number of cut_cycles, and p_2 = number of bounded regions. Thus Equation 7 yields the following expression:

$$N_{\mathrm{vertex}} - N_{\mathrm{edge}} + (N_{\mathrm{face}} - N_{\mathrm{a\_loop}}) = N_{\mathrm{a\_shell}} - N_{\mathrm{cut\_cycle}} + N_{\mathrm{region}} \quad (8)$$

Here, a *cut_cycle* is a cycle on a cell complex that cannot contract to a point. An *a_loop*, the abbreviation of "additional loop," stands for a loop that is not a primary loop of its face. If two loops belong to the same face, one of



8 Euler operations include (a) make_Edge_Vertex, (b) make_Edge_Face, and (c) make_Edge_Corner.

them is treated as a primary loop and the other as an a_loop. An *a_shell*, analogous to an a_loop, is a secondary shell of a region. Figure 5 illustrates some examples of a_loops and a_shells.

Boundary and neighborhood constraints

Equation 8 implies a constraint among primitive entities and bounding entities (vertices, edges, faces, loops, shells, and regions), but no constraints on coupling entities (ends, sides, corners, and feathers). Now, we return to Figure 2 to examine those kinds of constraints. Figure 2a depicts a neighborhood of a point in a region. Every point in the neighborhood, a ball of infinitesimal radius, belongs to the same region as the original point. No coupling entity exists within the neighborhood of a region point.

Figure 2b shows a neighborhood of a point on a face. The neighborhood of a face point consists of three portions: a disk of the face itself and two half-balls corresponding to the sides. This indicates that every face has two sides. This relation can be written as

$$N_{\text{side}} = 2\,N_{\text{face}} \tag{9}$$

Figure 2c illustrates a neighborhood of an edge point. The neighborhood of an edge point consists of blades and wedges as well as the edge itself; the number of blades is the same as the number of wedges. We can express this relation as

$$N_{\text{blade}} = N_{\text{wedge}} \tag{10}$$

Figure 2d shows a neighborhood of a vertex neighborhood. If we concentrate on the surface of the ball of infinitesimal radius instead of its inside, we can observe vertices, edges, and faces on the sphere. Since all of these vertices, edges, and faces are on a sphere, they satisfy the following equation:

$$N_{\text{vertex}} - N_{\text{edge}} + (N_{\text{face}} - N_{\text{a\_loop}}) = 2$$

Here, a vertex on the sphere corresponds to an end, because the vertex on the sphere is the intersection of an edge with a sphere of infinitesimal radius having its center on the original vertex. Similarly, an edge on the sphere corresponds to a fan, a face on the sphere corresponds to a corner, and an a_loop on the sphere corresponds to an *a_disk*, a secondary disk in a corner akin to an a_loop on a face. Therefore, we obtain the following equation as a constraint on a vertex neighborhood:

$$N_{\text{end}} - N_{\text{fan}} + (N_{\text{corner}} - N_{\text{a\_disk}}) = 2\,N_{\text{vertex}} \tag{11}$$

Focusing on neighborhoods of primitive entities produced Equations 9 through 11. Let us examine boundaries next. Since a vertex has no boundary, there is no constraint on a vertex boundary. An edge, however, has two ends as its boundaries. This constraint on edge boundaries can be expressed as

$$N_{\text{end}} = 2\,N_{\text{edge}} \tag{12}$$

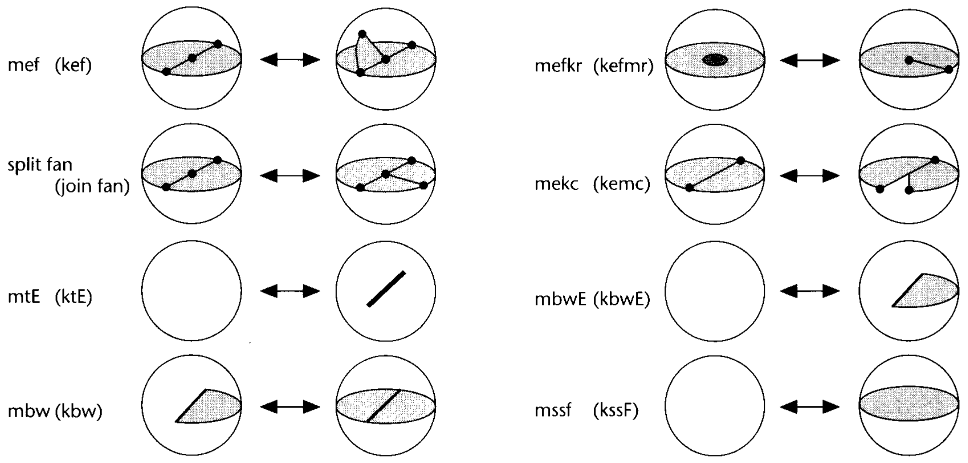
A face boundary constraint is a constraint on a loop cycle. The number of fans and blades on loops is the same. This constraint is expressed as

$$N_{\text{fan}} = N_{\text{blade}} \tag{13}$$

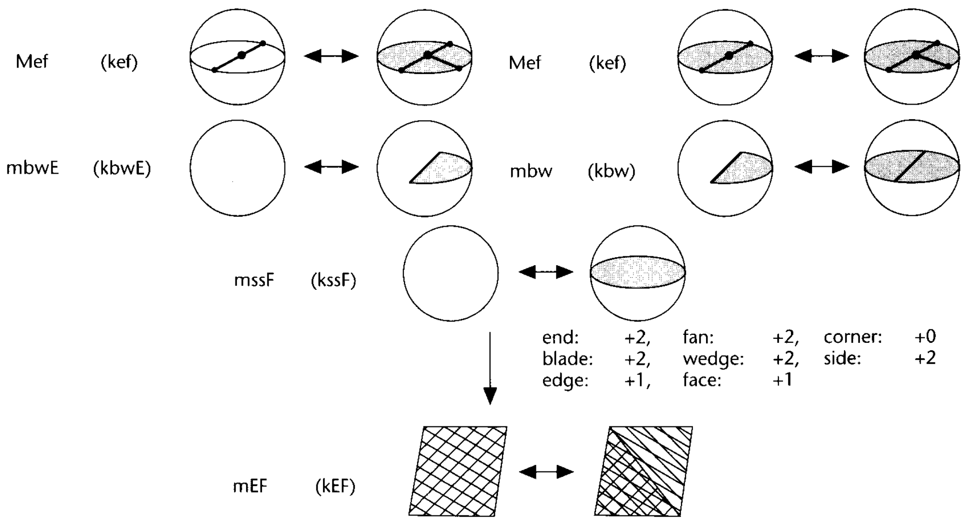
In the case of region boundaries, the arguments are rather complicated. However, we can obtain the following equation by the analogy of the Euler-Poincaré formula for a manifold solid. Each shell composed of corners, wedges, and sides can be seen as a two-manifold. The only differences are treatments of a_disks and a_loops.

$$(N_{\text{corner}} - N_{\text{a\_disk}}) - N_{\text{wedge}} + (N_{\text{side}} - 2\,N_{\text{a\_loop}}) = 2\,(N_{\text{shell}} - N_{\text{cut\_cycle}}) \tag{14}$$

Equations 9 through 14, obtained by examining neighborhoods and boundaries of primitive entities, provide essential constraints on nonmanifold topology in E^3. The equations must yield Equation 8, the interpretation of the Euler-Poincaré formula, by eliminating the terms of coupling entities. This means the equations can fulfill the constraint of the Euler-Poincaré formula by satisfying all constraints on boundaries and neighborhoods.



9 Neighborhood operations allow our system to manipulate all types of coupling entities.



10 Our system can build an Euler operation from neighborhood operations.

Euler and neighborhood operations

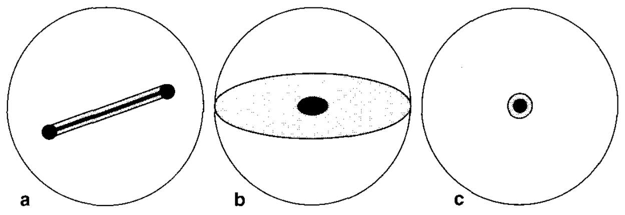
Euler operations are a class of operations that guarantees fulfilling the interpretation of the Euler-Poincaré formula (Equation 8). The equation contains variables standing for the numbers of vertices, edges, faces, regions, a_loops, a_shells, and cut_cycles. Thus, it gives a criterion for manipulating only those data entities. Figure 8 illustrates some examples of Euler operations with their short names. For instance, an operation “mEV” is an abbreviation of “make_Edge_Vertex”.

Neighborhood operations are a lower class of operations that maintain satisfaction of Equations 9 through 11. Since these equations include the variables standing for the numbers of all types of coupling entities, the neighborhood operations can manipulate all of the coupling entities. Some examples of neighborhood operations appear in Figure 9. In this figure, “mef” denotes “make_end_fan”.

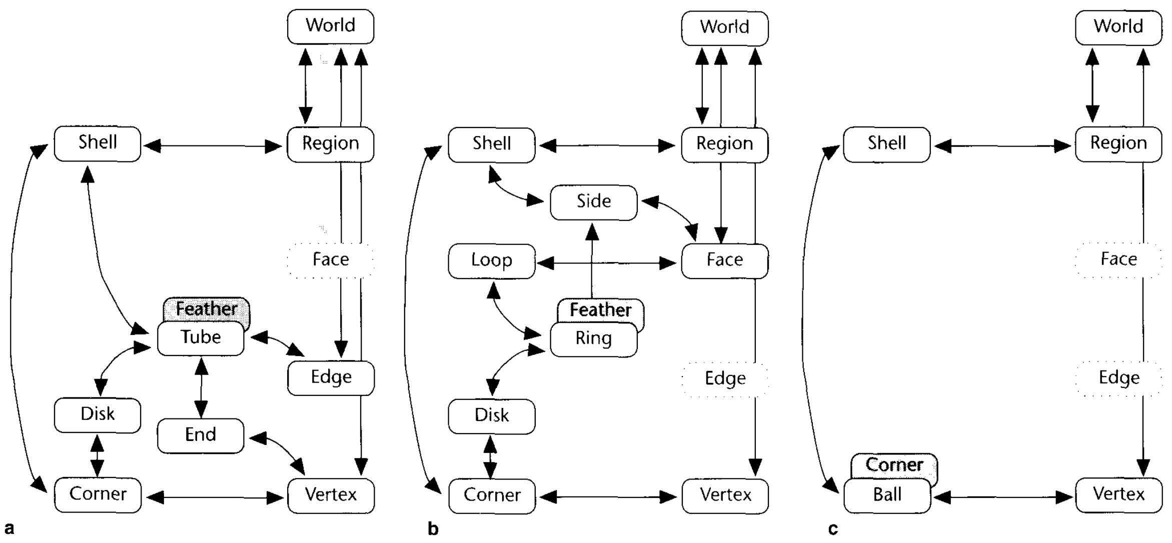
We have already explained how the constraint based on the Euler-Poincaré formula is derived from the constraints on boundaries and neighborhoods. Thus, an Euler operation can be composed of several neighborhood operations, all satisfying Equations 12 through 14. Figure 10 shows an example of how to implement an Euler operation with neighborhood operations. The Euler operation mEF(make_Edge_Face) is implemented as a composition of neighborhood operations: two mef(make_end_fan), one mbwE(make_blade_wedge_Edge), one mbw(make_blade_wedge), and one mssF (make_side_side_Face). At the same time, the variations of data entity numbers satisfy Equations 12 through 14.

Singularities

We defined the data structure and operations based on constraints. For instance, we derived the concept of a feather from the constraints on cycles in Equations 10 and 13. Neighborhood operations are those that generally maintain satisfaction of Equations 9 through 11. However, as Figure 11 (next page) illustrates, some singular cases do not satisfy those equations.



11 Our system produces only three singular cases of coupling entities: (a) the dangling edge, (b) the isolated vertex on a face, and (c) the isolated vertex in a region.



12 Data structures for the singular cases (a) tube, (b) ring, and (c) ball produce the new coupling entities.

Figure 11a shows a neighborhood of a point on a dangling edge. An incident region surrounds the edge, but no face is incident to it. The edge neighborhood consists of the edge itself and a regional portion. According to the definition of coupling entities, this portion corresponds to a wedge, which stands for the incidence relation between an edge and a region. But no blade exists in this edge neighborhood because the edge has no incident face. Therefore, Equation 10 is not satisfied. This means that a dangling edge does not have a regular radial cycle, so we cannot call the regional portion of the dangling edge neighborhood a wedge. To solve this dilemma, we introduced a new type of coupling entity called a *tube*, which represents an incidence relation between a region and a dangling edge.

A similar kind of singularity occurs at an isolated vertex on a face. The vertex is completely surrounded by the face, and no edge reaches the vertex (see Figure 11b). The vertex forms a loop. However, the loop does not have a regular loop cycle. An incidence relation exists between the vertex and the face, but no edge takes part in the loop. In other words, there is a fan but no blade in this loop cycle. We cannot treat the incidence relation between the vertex and the face as a fan without violating the constraint on a loop cycle, Equation 13. We created a new coupling entity, a *ring*, to represent this incidence relation.

An isolated vertex in a region causes a third kind of singularity. Only one region is incident to the vertex, as Figure 11c shows. Equation 11 is not satisfied in this case, because there are no ends, no fans, and no a_disks around the vertex. Thus, we cannot call the incident relation between an isolated vertex and the surrounding region a corner. To cope with this kind of singularity, we introduced a new coupling entity called a *ball*. Equations 15 and 16 replace Equations 11 and 14 in this case.

$$N_{\mathrm{end}} - N_{\mathrm{fan}} + (N_{\mathrm{corner}} - N_{\mathrm{a\_disk}}) = 2\,(N_{vertex} - N_{\mathrm{ball}}) \qquad \textbf{(15)}$$

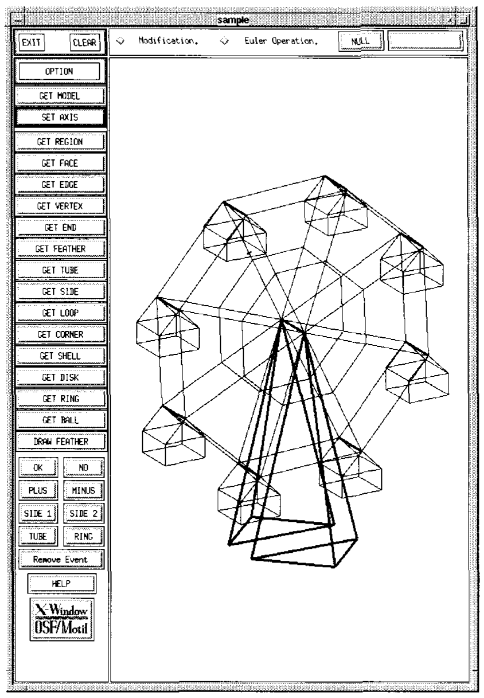
$$(N_{\mathrm{corner}} - N_{\mathrm{a\_disk}}) - N_{\mathrm{wedge}} + (N_{\mathrm{side}} - 2\,N_{\mathrm{a\_loop}}) = 2\,((N_{\mathrm{shell}} - N_{\mathrm{ball}}) - N_{\mathrm{cut\_cycle}}) \qquad \textbf{(16)}$$

These are the only modifications of the equations we need to make. Thus, the introduction of new coupling entities does not cause many variations of neighborhood operations.

Figure 12 shows the diagrams of the data structures for singular cases. You can see the characteristics of the singular coupling entities by comparing Figure 12 with the regular case in Figure 7. According to the definition, a tube resembles a wedge, which is represented by a pair of feathers in a regular case. The data structure reflects the similarity: Both have pointers to an edge and a disk. However, a tube does not have the pointer to a loop that a feather does. The data structure also indicates that we can see a ring as a feather without a pointer to an edge and a ball as a corner without disks.

Implementation

We implemented a prototype system on Unix work-



13 **The prototype system represented the incidence relations of this figure's regular lines with feathers and the bold lines with tubes.**

stations and experimented with the system. Figure 13 shows a sample output. The edges shown by bold lines have tubes representing their incidence relations. Otherwise, edges are regular and thus have feathers. The system was written in C++, which has two particularly advantageous features for our system.

The most important feature of C++ is the object-oriented concept. Every data entity is implemented as an object in C++. A Base class is a fundamental class: Every other class is derived from it (Figure 14). Every object has its own type and its own identifier because of inheriting the Base class, set at the time of the class' instantiation. It can be referred to at any time, but not modified.

An edge has either a radial cycle or a tube. This kind of alternative case is represented by union members feth, tube, and dummy. The member feth is a pointer to one of the feathers belonging to the radial cycle, if the edge is a regular one. The member tube points to a tube when an edge is of the dangling edge type. The last member, dummy, is a pseudopointer with which the system examines the type of the entity pointed to.

The other important feature is *function overloading*. Variations for "mEV(make_Edge_Vertex)" shown in Figure 8 differ in their roles and applicability. The top example in Figure 8a makes a dangling edge; the second example in Figure 8a also makes a dangling edge, but only when the original vertex is of the ball type. The third example in Figure 8a makes an edge on a face. The bottom example in Figure 8a makes an edge on a face as well, but using a ring-type vertex. Thus, the mEV functions should be defined individually, for which unction overloading is suited. Figure 15 shows the

```
class Base {
      Type    typ;
      unsigned int  id;
    public:
      Base(Type t, unsigned int i)
          {typ = t; id = i;};
      int ident() {return id;};
      Type type() {return typ;};
};

class Edge    : public Base {
      World* wrld;
      Edge*next;
      Edge*prev;
      union {
        Feather* feth;
        Tube*tube;
        Base*dummy;};
      static unsigned int number;
    public:
      Edge(World* w);
      // Member function
declarations       //(omitted)
};

    Edge::Edge(World* w) :
              (t_edge, number++) {
         addlist(w);
         wrld = w;
         dummy = (Base*)NULL;
    };
```

14 **Examples of class definitions for the (a) Base and (b) Edge classes, plus a constructor definition with initialization.**

```
overload mEV;
Edge* mEV(Corner* oldc, switch dir);
Edge* mEV(Ball* oldb, switch dir);
Edge* mEV(Feather* oldf, switch dir);
Edge* mEV(Ring* oldr, switch dir);3
```

15 **This example of function overloading creates four functions of the Euler operation make_Edge_Vertex.**

C++ declaration of the Euler operation mEV. With these four declarations, C++ creates four different functions having the same name mEV corresponding to each mEV in Figure 8a. The only difference between the four functions is the type of their arguments; C++ automatically assigns a relevant function according to the first argument.

Figure 16 on the next page shows an example of a function definition for one of the simplest definitions of the Euler operation mEV. The Euler operation consists of three neighborhood operations, mtE(make_tube_Edge), mecV(make_end_corner_Vertex), and med(make_end_

16 This function definition of mEV makes an edge and a vertex with a tube, a corner, an a_disk, and two ends.

```
Edge* mEV(Corner* oldc, Switch dir){
   Edge* newedge = lmtE(oldc->shl());
   lmecV(newedge->tub(), dir);
   lmed(oldc, newedge->tub(),
      (dir == PLUS) ? MINUS : PLUS);
   return newedge;
};
```

disk). It makes an edge and a vertex with a tube, a corner, an a_disk, and two ends. This change maintains satisfaction of Equations 12 through 14. The code is both simple and comprehensive.

Conclusions

Our study proposed a data structure and operations for nonmanifold topology based on neighborhood classification, as well as boundary representation. Our system has certain advantages. It clarifies the definitions of data entities, especially of additional data entities that represent incidence relations; the system also defines the singular cases clearly. It suggests the pointer set for representing incidence-ordering; the proposed set is sufficient, efficient, and convenient. It proposes a new class of operations named neighborhood operations, which can manipulate additional data entities and maintain neighborhood constraints. Finally, it implements an Euler operation as a composition of neighborhood operations satisfying boundary constraints. ■

Acknowledgments

This research work was partially supported by the Product Realization System's Development Project, organized by the Japan Society of Precision Engineering, and by project Q90-S66 under the Research Institute of Technology, Tokyo Denki University.

References

1. E. Brisson, "Representing Geometric Structures in *d* Dimensions: Topology and Order," *Proc. 3rd Symp. on Computational Geometry*, ACM Press, New York, 1987, pp. 218-227.
2. E.L. Gursoz, Y. Choi, and F.B. Prinz, "Vertex-Based Representation of Nonmanifold Boundaries," in *Geometric Modeling for Product Engineering*, M.J. Wozny, J.U. Turner, and K. Preiss, eds., North-Holland, Amsterdam, 1990, pp. 107-130.
3. K.J. Weiler, *Topological Structures for Geometric Modeling*, doctoral dissertation, Rensselaer Polytechnic Inst., Troy, N.Y., 1986.
4. D.P. Dobkin and M.J. Laszlo, "Primitives for the Manipulation of Three-Dimensional Subdivisions," *Proc. 3th Symp. on Computational Geometry*, ACM Press, New York, 1987, pp. 86-99.
5. P. Lienhardt, "Subdivisions of *N*-Dimensional Spaces and *N*-Dimensional Generalized Maps," *Proc. 5th Symp. on Computational Geometry*, ACM Press, New York, 1989, pp. 228-236.
6. K.J. Weiler, "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments," *IEEE CG&A*, Vol. 5, No. 1, Jan. 1985, pp. 21-40.
7. B. Baumgart, "Winged-Edge Polyhedron Representation," A.I. Report No. CS-320, Stanford Univ., 1972.
8. I.C. Braid, R.C. Hillyard, and I.A. Stroud, "Stepwise Construction of Polyhedra in Geometric Modeling," in *Mathematical Method in Computer Graphics and Design*, Academic Press, San Diego, Calif., 1980, pp. 123-141.
9. M. Mäntyla, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Md., 1988.
10. H. Masuda et al., "A Mathematical Theory and Applications of Nonmanifold Geometric Modeling," in *Advanced Geometric Modeling for Engineering Applications*, North-Holland, Amsterdam, 1990, pp. 89-103.
11. Y. Yamaguchi, K. Kobayashi, and F. Kimura, "Geometric Modeling with Generalized Topology and Geometry for Product Engineering," in *Product Modeling for Computer Aided Design*, North-Holland, Amsterdam, 1991, pp. 97-115.

Yasushi Yamaguchi *is an associate professor in the Department of Graphic and Computer Science of the University of Tokyo. His research interests lie in computer-aided design and geometric modeling, including parametric modeling, topology models for B-reps, surface-surface intersection, and surface interrogation. He received a BE in precision machinery engineering and a DrEng in information engineering from the University of Tokyo in 1983 and 1988, respectively. He is a member of ACM, Siggraph, IEEE Computer Society, and SIAM.*

Fumihiko Kimura *is a professor in the Department of Precision Machinery Engineering of the University of Tokyo. He received a BS in aeronautic engineering in 1968 and a DrEngSci in aeronautics in 1974, both from the University of Tokyo. His current research interests include the basic theory of CAD/CAM and CIM, concurrent engineering, engineering simulation, and virtual manufacturing. He is involved in the product model data exchange standardization activities of ISO/TC184/SC4. Kimura is a member of IFIP WG5.2 and 5.3, and a corresponding member of CIRP.*

Contact Yamaguchi at the Department of Graphic and Computer Science, University of Tokyo 3-8-1, Komaba, Meguro-ku, Tokyo 153, Japan, e-mail yama@komaba.c.u-tokyo.ac.jp.