Partial Entity Structure: A Compact Non-Manifold Boundary Representation Based on Partial Topological Entities

Sang Hun Lee
Graduate School of Automotive Engineering
Kookmin University, Korea
+82-2-910-4835
shlee@kookmin.ac.kr

Kunwoo Lee
School of Mechanical and Aerospace Engineering
Seoul National University, Korea
+82-2-880-7141
kunwoo@snu.ac.kr

ABSTRACT

Non-manifold boundary representations have gained a great deal of popularity in recent years and various representation schemes have been proposed because they allow an even wider range of objects for various applications than conventional manifold representations. However, since these schemes are mainly interested in describing sufficient adjacency relationships of topological entities, the models represented in these schemes occupy too much storage space redundantly although they are very efficient in answering queries on topological adjacency relationships. Storage requirement can arise as a crucial problem in models in which topological data is more dominant than geometric data, such as tessellated or mesh models.

To solve this problem, in this paper, we propose a compact non-manifold boundary representation, called the *partial entity structure*, which allows the reduction of the storage size to half that of the radial edge structure, which is known as a time efficient non-manifold data structure, while allowing full topological adjacency relationships to be derived without loss of efficiency. This representation contains not only the conventional primitive entities like the region, face, edge, and vertex, but also the *partial topological entities* such as the partial-face, partial-edge, and partial-vertex for describing non-manifold conditions at vertices, edges, and faces. In order to verify the time and storage efficiency of the partial entity structure, the time complexity of basic query procedures and the storage requirement for typical geometric models are derived and compared with those of existing schemes. Furthermore, a set of the generalized Euler operators and typical high-level modeling capabilities such as Boolean operations are also implemented to confirm that our data structure is sound and easy to be manipulated.

Keywords: Data Structure, Boundary Representation, Non-manifold, Topological Entity, Geometric Modeling

Solid Modeling 01 Ann Arbor Michigan USA

1. INTRODUCTION

As a key element in CAD/CAM applications, geometric modeling systems have evolved to widen their representation domains from wireframes to surfaces, to solids, and finally to non-manifold objects. Unlike previous geometric modelers, the non-manifold modeler allows any combination of wireframe, surface, solid, and cellular models to be represented and manipulated in a unified topological representation. This characteristic provides many advantages over conventional modelers as follows:

- The non-manifold modeler can provide an integrated environment for the product development process, as the non-manifold topological representation can represent different models required in various stages of product development; conceptual design, final design, analysis, and manufacture [5, 30]. For instance, abstracted models for conceptual design, mixed dimensional shapes for intermediate design steps, solid models for final design, mesh models on abstracted part shape for engineering analysis [26, 27], offset polyhedral models for tool path generation, and so on. Such a wide variety of geometric models can be created and manipulated by a non-manifold modeler, while previous modelers can represent only limited types of geometric models.
- Boolean operations are closed in the representation domain of non-manifold modelers unlike solid modelers [10, 11, 31]. The resultant shape of Boolean operations can be stored in a merged set, which contains not only the final Boolean result but also a complete description of the input primitives, all of the intersections between them, and historical information [7, 10, 23]. By using this merged set, B-rep models can be reshaped independently of their construction Boolean sequences [23], and a feature-based modeler based on B-rep can be easily implemented [23, 24].
- Traditional solid modeling functions like sweeping and offsetting operations can be applied to different dimensional objects. For instance, a sheet model is generated by sweeping wire edges, a solid model is generated by sweeping a sheet model, and a mixed dimensional model is generated by sweeping a mixture of sheets and wireframes [33]. In addition, a thin-walled solid model can be generated efficiently by sheet modeling and offsetting. In this approach, the mid-surface or one side of a thin plastic or sheet metal part is modeled as a sheet model at first, and then a thin-walled solid model is generated by applying the offsetting operation with a given thickness to this sheet model [15].

The research concerned with non-manifold modeling can be categorized into three groups: design of topological representation

schemes, specification of a set of primitive topological operators, and implementation of various high-level modeling functions like sweeping or Boolean operations. In the area of the design of topological representation schemes, several data structures such as the radial edge structure and the vertex-based representation have been suggested so far. These representations mainly focus on describing the sufficient adjacency relationships between topological entities in a non-manifold model without considering the storage size. As a result, although they are quite efficient for topological queries, they are so redundant and complicated that the models occupy too much storage space. The storage requirement can arise as a critical problem particularly for models in which topological data storage is more dominant than geometric data, such as tessellated or mesh models. Therefore, it is desirable to devise a new representation scheme that is more compact but is as efficient as the existing schemes.

To fulfill this requirement, in this paper, we propose a compact non-manifold boundary representation, called the *partial entity structure* (PES). This representation allows the reduction of the storage size to almost half that of the radial edge structure, which is one of the most popular and efficient of existing structures, while still allowing full topological adjacency relationships to be derived without loss of efficiency. To verify this improvement, the storage requirements of our representation for typical geometric models are estimated and compared with other existing representations, and the time complexities of all basic query procedures are investigated and compared.

Along with the data structure, considerable effort has also been made to design a set of primitive topological operators similar to the Euler operators in B-rep solid modelers [13, 14, 18, 19]. The Euler operators guarantee the integrity of the model because they were designed to make the model satisfy the Euler-Poincare formula for solid objects [20, 21]. These operators are also useful for insulating geometric modelling functionality of higher level from the specifics and complexities of underlying data structures. In addition, they allow easy implementation of undo commands, as each operator has its inverse operator. To take advantage of these useful features, several researchers have derived different Euler-Poincare formulae generalized for three-dimensional non-manifold models, and defined a set of Euler operators on the basis of their own formula.

In this paper, we suggest a set of basic topological operators based on a non-manifold Euler-Poincare formula proposed by Yamaguchi and Kimura [36]. Not only a minimal set of the Euler operators but also a practically sufficient set of the operators are proposed to enable efficient implementation of high-level modeling capabilities. Since these sets have been selected and examined through the development of a non-manifold modeler having various high-level modeling capabilities such as Boolean and sweeping operations, it guarantees the sufficiency and efficiency for practical implementation of modeling operations. Based on our proposed non-manifold data structure and Euler operators, an object-oriented non-manifold geometric kernel system has been developed.

The rest of this paper is organized as follows: Section 2 describes the previous work on non-manifold data structures. Section 3 represents an approach to measure the time and storage efficiency of a data structure, and our method to design a more optimal data structure based on this measurement. Section 4 describes the partial topological entities that are introduced to represent non-manifold conditions in a storage-efficient way. Section 5 shows the schematic diagram and the C++ implementation of our data structure. Sections 6 and 7 analyse and compare the time and storage complexity of our structure with those of other existing structures, especially the radial edge structure. Section 8 proposes a set of Euler operators generalized for *3*-D non-manifold models. Section 9 presents our conclusions.

2. RELATED WORK

The first significant work on the non-manifold B-rep was done by Weiler [31]. Weiler proposed an edge-based data structure called the radial edge structure (RES). In order to represent the adjacency relationships under non-manifold conditions at vertices, edges, and faces, he introduced face-use, loop-use, edge-use, and vertex-use topological entities that are associated with the face, loop, edge, and vertex entities, respectively. In virtue of these use entities, each region has its complete description of its boundaries just like a solid model in a solid boundary representation. Furthermore, in addition to a cyclic list of the edge-uses in a loop, a cyclic list of the face-uses adjacent to an edge is stored in the RES. According to Yamaguchi and Kimura [36], the ordering information of vertices and edges within a face is called a *loop cycle*, the ordering of faces and regions about an edge is a *radial cycle*, and ordering of edges and faces about a vertex is a *disk cycle*. Therefore, the RES represent explicitly the radial and loop cycles. The ordering information has been used in a lot of algorithms in computational geometry. In non-manifold modeling, by using the radial cycle information, a face-use can propagate to the incident face-uses over the edge-uses to form a shell bounding a closed region. However, in the RES, it is impossible to form a correct shell using only topological data when a non-manifold vertex has to be traversed, which is a vertex adjacent to two or more two-manifold surfaces or wire edges. This is because the RES does not keep any significant inclusion relationships between the incident surfaces to a non-manifold vertex. Although Weiler had already recognized this problem, he determined not to keep this information in the RES because it costs too much to maintain such information at each topology manipulation. Instead, he adopted a method in which the system extract this information using geometric and topological data whenever necessary.

To overcome this drawback of the RES, Choi proposed the Vertex-based Boundary Representation (VBR), in which the zone and disk topological entities are introduced to represent the inclusive relationships between the local regions at a vertex [6, 10]. In addition to the loop and radial cycles, the disk cycle is represented explicitly in the VBR. Thus, the VBR provides sufficient information for forming shells when a new region is created by adding a new face. However, Weiler's decision to derive such information using geometry data whenever necessary would be more rational from a practical view, because the zone and disk information is not frequently used in the normal geometric modeling process, and maintaining such information at every topology manipulation requires very high cost.

Yamaguchi and Kimura introduced six coupling entities to represent the neighborhoods and boundaries of basic topological entities, and suggested a data structure based on the coupling entities [36]. This representation also contains the three cycles mentioned above, and has the capability of providing as much topological information as the VBR. However, the resulting data structure looks equivalent to the VBR because they introduced the feather, which is not a coupling entity and has the same role as the cusp in the VBR, and excluded several coupling entities that

appear only in their representation, such as fans, blades, and wedges. For convenience, we will call this structure the Coupling Entity Structure (CES) in the rest of this paper.

Rossignac and O'Conner proposed the Selective Geometric Complexes (SGC), which are composed of finite collections of mutually disjoint cells [25]. The cells are open connected subsets of n-dimensional manifolds which are generalized concepts of vertices, edges, faces, and regions. The SGC can represent an object that has more than three dimensions or incomplete boundaries. Since the data structure of the SGC is based on a simple incidence graph that has no ordering information unlike the data structures mentioned above, it does not enable easy computation of certain important properties (orientability, for instance), although the topological information contained in this type of model is sufficient [17].

On the other hand, several works have been done to define topological representations for subdivided *n*-manifolds. Brisson defines the 'cell-tuple structure' to represent the incidence and ordering information in a subdivided n-manifold [3]. Lienhardt defines the 'n-G-map' and the 'n-map', based on combinatorial maps [16, 17]. Hansen and Christensen proposed a hierarchical 'split-element' representation, called the hyper data structure [12]. These data structures include the 'quad-edge' data structure of Guibas and Stolfi [9] and the 'facet-edge' data structure of Dobkin and Laszlo [8] as special cases in dimensions *2* and *3*, respectively.

3. RESEARCH GOAL

Since the representative non-manifold representations such as the RES or the VBR mainly focus on describing sufficient topological adjacency relationships and ordering information in non-manifold geometric models, they occupy so much storage space even though they are quite efficient in answering topological queries and traversals. The storage requirement can arise as a critical problem particularly in tessellated or mesh models where topological data is more dominant than geometric data. The efficiency and the storage of a data structure have a trade-off relationship, and the optimality of the data structure depends on its application.

Woo pioneered the analytic measures for analyzing the performance of solid boundary representations [35]. The optimal data structure for a general or specific set of applications can be constructed on the basis of the measures. When considering the solid boundary representation whose topological entities are vertices, edges, and faces, there are over 500 data structure schemata. The schemata are categorized into eight storage classes according to the number of adjacency relations to be stored. The storage complexity of a data structure is measured using counting formulas, and the time complexity of a data structure is measured using a set of primitive queries and update routines. The schemata form a storage-time curve in the shape of the letter 'L'. A globally optimal data structure must be as near the origin of the L-curve as possible. If one wants to design a faster data structure for a specific set of applications, he should investigate the frequency of topological queries first, and then store the adjacency relations in order of frequency. By applying the measures, the fact was discovered that the winged-edge data structure [1] is very close to the globally optimal data structure. Furthermore, Woo claimed that a new data structure that has a lower storage and a faster time than the winged-edge structure was found, and named it the symmetric data structure. The symmetric data structure belongs to the storage class which stores four relations out of nine, so called the ${}_9C_4$ storage class, and is the fastest and the optimal structure in this class. The symmetric data structure stores F→E, E→V, V→E, and E→F relations. Here, S, F, L, E, and V denote shells, faces, loops, edges, and vertices, respectively. If the symmetric data structure includes loops and shells, it can be extended to store S→F, F→S, F→L, L→F, L→E, E→L, E→V, and V→E relations. However, we need to pay attention to the fact that the symmetric data structure is a sort of incidence graph, whereas the winged-edge structure represents ordered topological models.

There has been no research work for finding an optimal data structure for non-manifold models. However, the analytic measures proposed by Woo are also effective in non-manifold data structure design. Currently, the RES is used widely by researchers for non-manifold modeling and its applications. Not only commercial modelers such as SMLib of Solid Modeling Solutions [28], but also academic modelers for the research of various non-manifold applications have adopted the RES as their topological framework [4, 26]. However, it is also true that there has been little analytic rationalization of its time and storage efficiency by its users.

In non-manifold boundary representations, there are six basic topological entities: R, S, F, L, E, and V, and thus thirty-six adjacency relationships may be represented in the data structure. Here, R denotes regions. To facilitate the comparison between the symmetric data structure and the RES, let us consider only the adjacency relationships among R, F, E, and V. Then, the RES stores basically six relationships out of sixteen: R→F, F→R, F→E, E→F, E→V, and V→E. If a region is not bounded by any wireframe, the RES describes the region boundaries just as the symmetric data structure describes a solid boundary. Thus, we can conjecture that the RES is the fastest data structure in the ${}_{16}C_6$ storage class which it belong to, because a non-manifold model is composed of a set of regions each of which is described by the symmetric data structure that is the fastest in its storage class. However, the storage efficiency of the RES is not optimal, because the boundaries of regions are stored twice in different orientations. Therefore, if we reduce the storage size as much as possible while preserving the time efficiency of the RES, we can achieve a compact as well as time-efficient data structure.

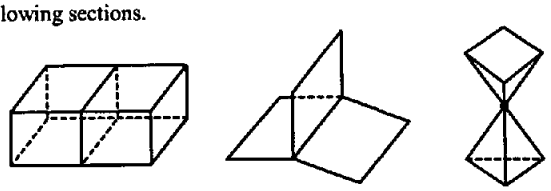
Regarding the zone and disk information, the RES does not include this information because it costs very much to maintain this information at every primitive topological operation while this information is rarely used in actual geometric modeling operations. Instead, it is driven from the topology and geometry data whenever necessary. If this information is omitted, both the VBR and the CES are equivalent to the RES.

The goal of this paper is to construct a new non-manifold boundary representation that requires less storage than the RES while still allowing full topological adjacency relationships to be derived with the same time efficiency as the RES. To achieve this goal, we introduce new topological entities for representing the non-manifold conditions in time- and storage-efficient ways, and design a new data structure based on these entities. In addition, to verify the compactness and the time efficiency of our representation, the storage and the time complexity of our representation and the RES are derived and compared with each other.

4. TOPOLOGICAL ENTITIES

The cell complex embedded in E^3 is widely adopted as a suitable mathematical model for 3-D non-manifold objects [8, 22, 23]. The cell complex of E^3 is a finite collection of *0*-, *1*-, *2*-, and *3*-cells: the *0*-cell is equivalent to a vertex, the *1*-cell to a edge, the *2*-cell to a face, and the *3*-cell to a region. Mathematically, an *n*-cell is defined as a bounded subset of E^n that is homeomorphic to an *n*-dimensional open sphere, and whose boundary consists of a finite number of lower dimensional cells. A cell complex of E^n is a finite collection K of cells of E^n, $K = \cup\{\alpha : \alpha \text{ is a cell}\}$ such that if α and β are two different cells in K, then $\alpha \cap \beta = \emptyset$. The cell complex of E^3 is accepted as a topological model of our representation, and it is intuitively a mixture of wireframes, surfaces, and solids, which we want to describe.

The topological entities in our boundary representation are classified into two large groups; the primary and the secondary entities. The primary topological entities consist of *0*- to *3*-cells (i.e., vertex, edge, face, and region) and their bounding elements (i.e., loop and shell). They are used commonly in 3-D boundary representations, and their definitions are exactly the same as those of the existing representations such as the RES, the VBR, or the CES. The secondary topological entities are the partial face (*p-face*), partial edge (*p-edge*), and partial vertex (*p-vertex*). They are called *partial topological entities* or *partial entities* all together. They are introduced to represent adjacency relationships among the primary entities. The p-face is used to represent the non-manifold condition where a face is adjacent to two regions as shown in Figure 1(a). The p-edge is introduced for the non-manifold condition where more than two faces are connected to an edge as illustrated in Figure 1(b). The p-vertex is for the non-manifold case where more than one two-manifold surfaces are connected to a vertex as shown in Figure 1(c). Now we will explain the partial entities along with the primary entities in the following sections.



(a) (b) (c)

Figure 1: Typical non-manifold conditions. (a) a face with two incident regions. (b) an edge with three incident faces. (c) a vertex with two incident two-manifold surfaces.

4.1 Partial Faces

In our representation, a model is the highest level of topological entity, which can be an object of manipulation in the non-manifold modeler. A model is composed of one or more regions: one infinite open region and *0* or more finite closed regions. For example, the box with an inner partition shown in Figure 1(a) is represented by a non-manifold model with three regions: an infinite external region R_0 and two closed internal regions R_1 and R_2 as illustrated in Figure 2. To distinguish a material-filled region from an empty one, an attribute is assigned to each region. It is set as *solid* for a filled region, and *void* for an empty region. The region is bounded by the oriented boundaries, the shells. A region has a single peripheral shell and *0* or more void shells. In Figure 2, S_0 is the virtual peripheral shell of the infinite region R_0, S_1 is a void shell of R_0, and S_2 and S_3 are the peripheral shells of R_1 and R_2, respectively. The normal of a shell is directed to the inside of the region as illustrated in Figure 2. However, if a single vertex or wireframe edges are associated with the shell, those portions are not oriented exceptionally.

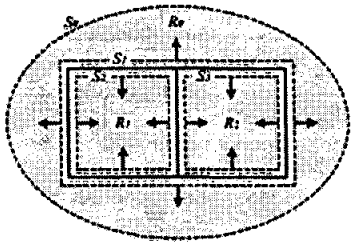


Figure 2: Regions and shells of the model in Figure 1(a).

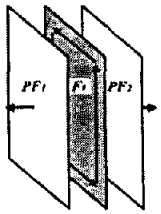


Figure 3: Example of partial faces in a face.

In a non-manifold model, a face bounds two incident regions, and thus each side of a face should be a part of the boundary of each region. To meet this requirement, we split a face into two p-faces. P-faces gather to compose a shell. This is similar to having two half edges for each edge to specify the boundary of a face in the half-edge data structure for solid objects [20]. As shown in Figure 3, a p-face is usually one side of the corresponding face. The normal of a p-face is directed to the inside of the region as illustrated in Figure 2 and Figure 3. However, there are exceptions when an isolated vertex or wireframe edges are included in a shell. In that case, an unoriented p-face is included for the consistency of the data structure. In the C++ implementation of the data structure, as shown in Figure 7, a pointer to an isolated vertex or a wire edge is assigned to the *_child* field of the *Pface* class, and the *_orient* field is set as *unoriented*. The dotted lines (a) and (b) in Figure 6 indicate the exceptional cases of a single-vertex shell and a wire edge, respectively.

The concept of the p-face is similar to the face-use of the RES, the wall of the VBR, and the side of the CES. When comparing the p-face with the face-use, however, there is an important distinction between two topological entities. The face-use has its own boundaries in the form of loop-uses while the p-face does not include any boundaries. The faces of the RES, the VBR, and the CES have no data for their boundaries while the face of our representation has its own boundary data. Because the boundaries of a face-use are coincident with those of its mate face-use except the orientation, one of them is redundant. This argument is also effective to both the VBR and the CES, because the wall and the side are designed in the same manner with the face-use of the RES. However, in our representation, the boundary information of a p-face is derived from the boundary of the corresponding face considering the orientation of the p-face. By introducing the p-

face, the storage cost of our representation can be reduced to almost half that of the RES.

Another big distinction between the p-face and the face-use is that the p-face can be associated with not only a face but also a wire edge or an isolated vertex in a region, while the face-use is associated with only a face. Furthermore, a shell in the RES can include only one of the following mutually exclusive alternatives: a list of face-uses in a shell, a list of edge-uses for wire edges, and a vertex-use for a single-vertex shell. This fact implies that the boundary of a region composed of a mixture of lamina faces and wire edges cannot be represented directly by a single shell. In this case, such a region boundary should be represented by two shells for the faces and the wire edges. On the contrary, in our representation, any mixture of lamina faces and wire edges is directly represented by a single shell, because the p-face represents not only the use of a face but also the use of a wire edge or an isolated vertex by a region.

4.2 Partial Edges

The face is a surface bounded by a single peripheral loop and *0* or more hole loops. The loop is a connected and oriented boundary of a face, and its orientation follows the right hand rule with respect to the normal of the face geometry in our representation. Loops are classified into two types: peripheral and hole loops. The peripheral loop is the outer boundary of a face, while the hole loop is the inner boundary. The peripheral loop has a counter-clockwise direction with respect to the normal of the face geometry while the hole loop has a clockwise direction. If an isolated vertex exists in a face, an unoriented hole loop is assigned to the vertex. An edge is a bounded and open curve, which does not include its end points. In our representation, the edge is bounded by two p-vertices.

In a non-manifold model, an edge is adjacent to an arbitrary number of faces, while it always neighbors two faces in a two-manifold model. Since an edge should serve as the boundaries of the incident faces, we split an edge into as many p-edges as the incident faces. For instance, the p-edges in the non-manifold sheet model shown in Figure 1(b) are illustrated in Figure 4(a). Here, the edge E_1 is split into three p-edges PE_1, PE_2, and PE_3, for the three faces F_1, F_2, and F_3, respectively.

A p-edge is a component of a loop. A loop has a cyclic list of p-edges. In normal cases, a p-edge has an edge pointer and an orientation flag with respect to the edge geometry. As illustrated in Figure 4(a), the direction of a p-edge follows that of the corresponding loop, and its orientation flag is set accordingly. In this manner, the loop cycle is directly represented in our representation. In the case of a single-vertex loop, the p-edge is unoriented and points to the isolated p-vertex instead of an edge. In the schematic diagram of our representation shown in Figure 6, the dotted line (c) indicates this special case.

Along with the loop cycle, in our representation, the radial cycle of loops around an edge is represented by a cyclic list of the p-edges around the edge. As illustrated in Figure 4(b), the p-edges around an edge are ordered following the right hand rule with respect to the direction of the edge geometry. In order to facilitate the search of p-edges in the reverse direction, the p-edges of an edge are stored using a doubly linked list. This radial cycle information is very useful for searching for a group of p-faces forming a new shell when a region is separated in two. Figure 7 illustrates the implementation of the class for the p-edge in C++ language.

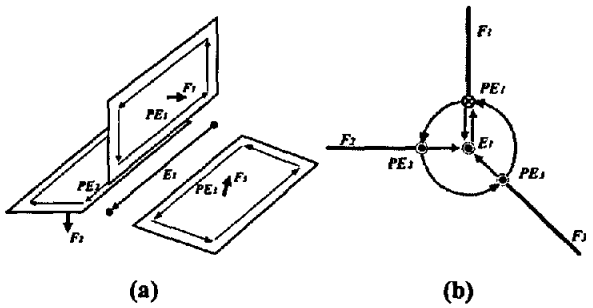


Figure 4: Partial edges in loops and edges. (a) partial edges ordered in the corresponding loop. (b) partial edges ordered in the corresponding edge.

The p-edge is distinguished from the edge-use of the RES by its definition and usage. The p-edge in our representation composes a loop bounding a face, while the edge-use composes a loop-use bounding a face-use that is one side of a face. Since a face has two face-uses in the RES, the number of edge-uses around an edge is twice that of p-edges. The discussion about the VBR and the CES can be done in the same manner, because the cusp of the VBR and the feather of the CES are equivalent to the edge-use of the RES.

Incidentally, the p-edge looks similar to the co-edge of ACIS [29], which is a commercial solid modeling kernel of Spatial Technology Inc., because a co-edge is given to each incident face of the edge. However, the p-edge is distinguished from the co-edge by the historical background and usage. The p-edge is designed for non-manifold boundary representation from the beginning, while the co-edge is initially introduced as the half-edge of the half-edge data structure for solid modeling and then extended to accommodate the non-manifold condition at an edge. The p-edge is associated only with the edges or isolated vertices bounding a face, while the co-edge can be associated even with wire edges. This seems to be a heritage of the solid boundary representation where a wire edge is associated with two co-edges to form a loop. In solid modeling, when a wireframe is closed by adding an edge, a new face should be created according to the Euler-Poincare formula. In our representation, a wire edge is directly associated with a p-face to form a shell. Anyway, this similarity gives our representation a great advantage over the RES when the representation of ACIS is migrated to our representation. Just by introducing the p-face and the p-vertex, the ACIS representation can be converted to an immaculate non-manifold boundary representation. The discussion about Parasolid can be done in the same manner, as the fin of Parasolid is equivalent to the co-edge of ACIS.

4.3 Partial Vertices

In a non-manifold model, a vertex can be adjacent to an arbitrary number of two-manifold surfaces. Note that a two-manifold surface is formed by a group of connected faces, and a wire edge can be dealt with as a degenerate case of a surface. We introduce the p-vertex in order to handle such a non-manifold condition at a vertex. The readers may imagine that the p-vertices for a vertex are formed by splitting the vertex into as many pieces as the adjacent surfaces. As illustrated in Figure 5, each p-vertex is a linkage to a surface or a wire edge. While the two-manifold vertex is associated with only one p-vertex, a non-manifold vertex can be associated with more than one p-vertices.

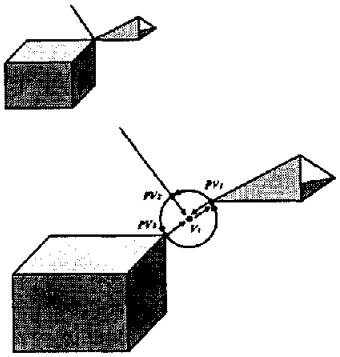


Figure 5: Example of partial vertices around a vertex.

As illustrated in Figure 7, the p-vertex class has two record fields: one is the pointer to the parent vertex, and the other is the pointer to an edge associated with a two-manifold surface or a wire edge. However, if an isolated vertex forms a loop, that is, in the case of a single-vertex loop, the p-vertex points to the single p-edge of the loop instead of an edge. This exceptional case is represented as the dotted line (c) in the schematic diagram of our representation shown in Figure 6.

Since a p-vertex points to only one edge out of the connected edges, the algorithms for searching for various adjacent topological entities need to be developed. In Section 6.1, an algorithm is introduced for finding all of the edges connected to a p-vertex. By applying this algorithm, all query functions searching for the entities adjacent to a given p-vertex can be implemented. The disk information of the VBR and the CES can also be extracted using the orientations of the p-edges and p-faces around a p-vertex.

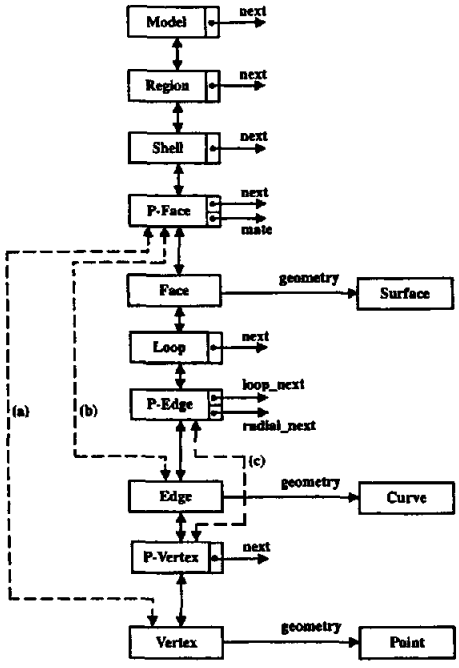


Figure 6: Schematic diagram of the partial entity structure.

5. DATA STRUCTURE

The hierarchical data structure to store these topological entities with their relationships is illustrated in Figure 6. The data structure is composed of the topology and geometry parts like other traditional B-reps. This data structure is named the *Partial Entity Structure* (PES) in this paper. In Figure 6, each arrow represents a pointer to another entity. The pointers drawn with dotted arrows are used to describe three exceptional cases: single-vertex shells, single-vertex loops, and wire edges, while the pointers drawn with vertical solid arrows serve for normal cases. Here, a single-vertex shell represents a shell containing only a vertex with no adjacent edges or faces, a single-vertex loop represents a loop including only a vertex with no adjacent edges, and the wire edge represents an edge that does not have any incident face.

The implementation of our data structure with the C++ classes is described in Figure 7. Each parent node point to one of its child nodes while all child nodes point to their parent node. All the sibling entities are singly linked except p-edges in order to save storage space. P-edges are doubly linked and ordered for the associated loop and edge. Note that the *_next* fields in the face, edge, and vertex are introduced only in order to enhance the efficiency of their traversal through the model for display or picking even though they are redundant.

```
class Entity {
  int        _id;
  Attribute  *_attribute;
};
class Model : public Entity {
  Model    *_next;          // next model
  Region   *_region;        // list of regions
};
class Region : public Entity {
  Region   *_next;          // link field of the region list of a model
  Model    *_model;         // parent model
  Shell    *_shell;         // peripheral shell
};
class Shell : public Entity {
  Shell    *_next;          // next void shell
  Region   *_region;        // parent region
  Pface    *_pface;         // partial face
};
class Pface : public Entity {  // partial face (p-face) class
  Pface    *_next;          // next p-face
  Shell    *_shell;         // parent shell
  Entity   *_child;         // child entity: a face, an edge, or a vertex
  Orient   _orient;         // orientation flag w.r.t. the face normal
  Pface    *_mate;          // mate p-face
};
class Face : public Entity {
  Pface    *_pface;         // one of two incident p-faces
  Loop     *_loop;          // peripheral loop
  Surface  *_geometry;      // surface
};
class Loop : public Entity {
  Loop     *_next;          // next hole loop
  Face     *_face;          // parent face
  Pedge    *_pedge;         // a p-edge in a loop
};
class Pedge : public Entity {  // partial edge (p-edge) class
  Loop     *_loop;          // parent loop
  Entity   *_child;         // child entity: an edge or a p-vertex
  Orient   *_orient;        // orientation flag w.r.t. the edge direction
  Pvertex  *_pvertex;       // start p-vertex
```

```
  Pedge    *_looped_prev;  // previous p-edge in the loop cycle
  Pedge    *_looped_next;  // next p-edge in the loop cycle
  Pedge    *_radial_prev;  // previous p-edge in the radial cycle
  Pedge    *_radial_next;  // next p-edge in the radial cycle
};
class Edge : public Entity {
  Entity   *_parent;       // parent entity: a p-edge or a p-face
  Pvertex  *_pvertex[2];   // two end p-vertices
  Curve    *_geometry;     // curve
};
class Pvertex : public Entity { // partial vertex (p-vertex) class
  Pvertex  *_next;         // another p-vertex associated with _vertex
  Entity   *_parent;       // parent entity: an edge or a p-edge
  Vertex   *_vertex;       // mother vertex
};
class Vertex : public Entity {
  Entity   *_parent;       // parent entity: a p-vertex or a p-face
  Point    *_geometry;     // position
};
```

Figure 7: Implementation of the partial entity structure with the classes in C++.

6. ANALYSIS OF TIME COMPLEXITY

The time complexity of a data structure is usually evaluated by measuring the response time of each basic query that return a specific type of topological entities adjacent to a given entity. In this paper, we also adopted this method to compare the time efficiency of our structure with that of the RES. For this work, we estimate the time efficiencies of not only basic queries but also a shell search procedure that finds a set of connected p-faces from a given p-face. In non-manifold modeling, this procedure is used frequently for detecting separation of a region, or for forming a new shell by adding a new face. Our representation is also efficient in performing this procedure in virtue of keeping the radial cycle information around an edge.

Let R, S, F, L, E, V, Pf, Pe, Pv, Fu, Lu, Eu, and Vu denote the region, shell, face, loop, edge, vertex, p-face, p-edge, p-vertex, face-use, loop-use, edge-use, and vertex-use, respectively. If A and B represent one of the entities enumerated above, we will use the following symbols to facilitate the discussion:

A_i	an A entity
A, $\{A_i\}$ or $\{A_i\}^m$	a set of A entities, where m is the number of A entities
$<A_i>$ or $<A_i>^m$	an ordered set of A entities
BA_i	the number of B entities adjacent to an A_i entity
$A_i \rightarrow \{B_i\}$	retrieval of all B entities adjacent to an A_i entity
$A \rightarrow B$ or $\{A_i\} \rightarrow \{B_i\}$	retrieval of all B entities adjacent to all A entities

6.1 Basic Topological Queries

The basic topological entities used in existing non-manifold boundary representations are of six types: region, shell, face, loop, edge, and vertex. Hence, all possible adjacency relationships between two types of entities are counted to thirty-six as illustrated in the adjacency relationship matrix in Table 1. This matrix also briefly describes the algorithms of query procedures that collect a specific type of entities adjacent to a given entity. This matrix verifies that our representation can also provide the same adjacency relationships including the ordering information as the RES does. In our representation, nine upward and downward diagonal adjacency relationships boxed by solid lines are directly represented in normal conditions. Note that our representation does not directly store a list of all edges adjacent to a vertex, while the RES does. In our representation, a vertex has only a list of p-vertices, and each p-vertex points to only one of its adjacent edges. Therefore, a procedure to find edges adjacent to a p-vertex needs to be developed. The algorithm for $V_i \rightarrow \{E_i\}$ is introduced in the latter part of this section. In addition, six other adjacency relationships boxed by dotted lines are also represented directly under three special singular conditions: a single-vertex shell ($S \rightarrow V$ and $V \rightarrow S$), a single-vertex loop ($L \rightarrow V$ and $V \rightarrow L$), and a wire edge ($S \rightarrow E$ and $E \rightarrow S$). The remaining twenty-six adjacency relationships are derived from the existing adjacency relationship information. Since all thirty-six adjacency relationships for a non-manifold model can be derived from our representation, it can be said that our representation is complete according to Weiler's definition [31].

In order to evaluate and compare the time efficiencies of the PES and the RES, we need to measure the time required for executing each basic query function. The total running time of a query function is the sum of two times: one is the time for executing the instructions, and the other is the time for accessing the records and fields of a data structure. The number of record and field accesses is a more important criterion than the number of instructions, because a database access may cause a hard-disk

Reference Entity	*Adjacent Group Entity*					
	Regions	*Shells*	*Faces*	*Loops*	*Edges*	*Vertices*
Region	$\{\{R_i\}^{FSi}\}^{SR}$	$\{S_i\}^{SR}$	$\{\{F_i\}^{FSi}\}^{SR}$	$\{\{\{L_i\}^{LFi}\}^{FSi}\}^{SR}$	$\{\{\{<E_i>^{ELi}\}^{LFi}\}^{FSi}\}^{SR}$	$\{\{\{<V_i>^{ELi}\}^{LFi}\}^{FSi}\}^{SR}$
Shell	$\{R_i\}^1$	$\{S_i\}^{FSi}$	$\{F_i\}^{FSi}$	$\{\{L_i\}^{LFi}\}^{FSi}$	$\{\{<E_i>^{ELi}\}^{LFi}\}^{FSi}$	$\{\{<V_i>^{ELi}\}^{LFi}\}^{FSi}$
Face	$\{R_i\}^2$	$\{S_i\}^2$	$\{<<F_i>^2>^{ELi}\}^{LFi}$	$\{L_i\}^{LFi}$	$\{<E_i>^{ELi}\}^{LFi}$	$\{<V_i>^{ELi}\}^{LFi}$
Loop	$\{R_i\}^2$	$\{S_i\}^2$	$\{F_i\}^1$	$<<L_i>^2>^{ELi}$	$<E_i>^{ELi}$	$<V_i>^{ELi}$
Edge	$<R_i>^{LEi}$	$<S_i>^{LEi}$	$<F_i>^{LEi}$	$<L_i>^{LEi}$	$<<E_i>^2>^{LEi}$	$<V_i>^2$
Vertex	$\{R_i\}^{LFi}$	$\{S_i\}^{LFi}$	$\{F_i\}^{LFi}$	$\{L_i\}^{LFi}$	$\{E_i\}^{LFi}$	$\{V_i\}^{LFi}$

Table 1: The adjacency relationship matrix of the primary topological entities in the partial entity structure (PES).

access and at least requires a main memory access. In most of previous work, therefore, the number of database accesses of each basic query function is counted to evaluate and compare the time complexity of data structures. We also adopted this method to compare the efficiency of the PES with the RES. Now we will show how to calculate the time complexities of the selected queries, and summarize the results for all of the queries in the form of a table. Note that it is allowed to visit an entity more than once during the traversal in the basic query function unless otherwise stated.

Figure 8 shows the algorithm for a query procedure that retrieves the faces adjacent to a given region. For the sake of convenience, let us assume that the private members of C++ classes for topological entities can be accessed directly by query functions. Like this algorithm, most of the query procedures have nested loops. If SR_i and PfS_i denote the number of shells in a region and the number of p-faces in each shell respectively, the total number of field accesses N_f is:

$$N_f = \sum_{j=1}^{SR_i}(1+3PfS_j)$$
$$= SR_i + 3\sum_{j=1}^{SR_i} PfS_j$$
$$= SR_i + 3PfR_i.$$

As shown in the formula above, $1+3PfS_j$ times of field accesses occur for each shell. For each shell, as shown in Figure 8, the *_next* field of a shell is accessed once in line 1, and the *_next*, *_child*, and *_face* fields of the p-face are accessed as many times as the number of p-faces in a shell in lines 2 to 4. Therefore, the total number of field accesses for visiting all of the faces of a region is reduced to $SR_i + 3PfR_i$. Since the PfR_i term is much bigger than SR_i, the dominant term of this formula is $3PfR_i$. If FuR_i denotes the number of face-uses in a region in the RES, FuR_i is exactly the same as PfR_i. In order to facilitate the comparison between the PES and the RES, we will use FuR_i instead of PfR_i later.

```
FACES-IN-REGION (r, f_list)
1  for each shell s in r do
2    for each p-face pf in s do
3      if pf→_child_type is face
4        add pf→_face to f_list;
```

Figure 8: Query procedure for finding faces adjacent to a given region.

As mentioned above, the adjacency relationship $V \rightarrow E$ is not represented directly in our representation. A vertex has a set of p-vertices and each p-vertex has only a pointer to one of the edges connected to the p-vertex. Thus, it is necessary to develop an algorithm for finding all of the edges adjacent to a given vertex. The procedure for $V_i \rightarrow \{E_i\}$ using the p-edge information is described in Figure 9. Note that in this algorithm, each of the edges adjacent to a vertex is visited only once although multiple visiting is allowed to the query procedures according to the assumption in the beginning of this section. This is because the vertex of our representation does not have any listed information about its adjacent entities.

```
EDGES-OF-VERTEX (v, e_list)
1  for each p-vertex pv at v do
2    EDGES-OF-PVERTEX (pv, pv→_edge, e_list);
3  for each edge e in e_list do
4    mark e unvisited;

EDGES-OF-PVERTEX (pv, e, e_list)
1  add e to e_list;
2  mark e visited;
3  if e is a wire edge
4    return;
5  for each p-edge pe around e do {
6    if pv is the start vertex of pe
7      next_e = pe→_prev_in_loop→_edge;
8    else
9      next_e = pe→_next_in_loop→_edge;
10   if next_e is unvisited
11     EDGES-OF-PVERTEX (pv, next_e, e_list);
12 }
```

Figure 9: Query procedure for finding edges adjacent to a given vertex.

Suppose all edges in a data structure are initially marked unvisited. Marking can be stored in the storage for attributes. The procedure EDGES-OF-PVERTEX is invoked for each p-vertex of a given vertex to gather all of the edges adjacent to the vertex. The input arguments of EDGES-OF-PVERTEX are a p-vertex pointer *pv*, an edge pointer *e* of the p-vertex, and an edge list pointer *e_list*; and the output argument is *e_list* whose content includes the edges visited. The first task of this procedure is to mark the input edge *e* visited and add it into the edge list *e_list* for output. Then, every unvisited edge adjacent to *pv* is visited in turn using EDGES-OF-PVERTEX recursively. This is a typical depth-first search algorithm. The edge *next_e* to be visited next is determined using p-edges in a loop. Once all of the edges adjacent to the p-vertices around *v* have been visited, the search is complete and the marks of the found edges are reset as unvisited.

Now let us try to count the field and record accesses in this query procedure. The procedure EDGES-OF-PVERTEX is called exactly once for each edge adjacent to *v*, because it is invoked only for an unvisited edge, and its first task is marking the edge as visited. During an execution of EDGES-OF-PVERTEX, field accesses occur $5PePv_i+2EPv_i$ times, because there are two field accesses in lines 2~3, and $5PeE_i$ field accesses in the loop in lines 5~12. Therefore, during an execution of EDGES-OF-VERTEX, the total number of data structure field accesses is $5PeV_i+3EV_i+2PvV_i$ if the field accesses of *e_list*, which is not a part of the data structure, are ignored. The most dominant term of this formula is $5PeV_i$. The record accesses occur $3PeV_i+3EV_i+PvV_i$ times, and can be represented as $3PeV_i$ in the same manner. However, if the number of edges connected to a vertex is not large, PeV_i is not significantly greater than EV_i. For example, $PeV_i = 2EV_i$ under two-manifold conditions. Therefore, we should be careful in asserting that our query procedure is more efficient than that of the RES, although the most dominant term is a bit less than that of the RES. Actually, because the RES stores directly all of the edges adjacent to a vertex using vertex-uses, it is always faster than our representation in the following queries for vertices: $V_i \rightarrow \{R_i\}$, $V_i \rightarrow \{S_i\}$, $V_i \rightarrow \{F_i\}$, $V_i \rightarrow \{L_i\}$, $V_i \rightarrow \{E_i\}$, and $V_i \rightarrow \{V_i\}$ in Table 1. However, as illustrated in Tables 2 and 3, the order of time complexity of the vertex query procedures of the PES is the same as that of the RES.

We investigated the counts of the field and record accesses for all basic queries of our representation, and summarized them as a matrix in Table 2. The same process has been done for the RES and its result is summarized as a matrix in Table 3. The number outside the parentheses in each box denotes the number of field accesses in a query, and the figure in the parentheses denotes the number of record accesses. To facilitate the comparison, we selected more convenient counting variables between the RES's and the PES's based on the special relations such as $EuFu_i = PeF_i$, $EuLu_i = PeL_i$, $EuE_i = 2PeE_i$, and $EuV_i = 2PeV_i$. The reason why we use the counting variables of the RES for the adjacency relationships of shells and regions, such as LuR_i, LuS_i, EuR_i, *and* EuS_i, is that the corresponding variables of the PES cannot be used as correct counts if a shell includes laminar faces.

As appears in Tables 2 and 3, the order of time complexity of each query of the PES is exactly the same as that of the RES. The PES answers faster than the RES for 12 queries, slower for 19 queries, and at the same speed for 5 queries. As a result, it is proven that the PES has almost the same time efficiency as the RES.

	$\{R_i\}$	$\{S_i\}$	$\{F_i\}$	$\{L_i\}$	$\{E_i\}$	$\{V_i\}$
R_i	5(3) FuRi	4(2) FuRi	3(1) FuRi	1(1) LuRi	3(1) EuRi	4(2)EuRi
S_i	1(1)	4(2) FuSi	3(1) FuSi	1(1) LuSi	3(1) EuSi	4(2) EuSi
F_i	6(5)	4(3)	8(5) PeFi	1(1) LFi	3(1) PeFi	3(2) PeFi
L_i	7(6)	5(4)	1(1)	6(3) PeLi	3(1) PeLi	3(2) PeLi
E_i	8(6) PeEi	7(5) PeEi	3(2) PeEi	2(1) PeEi	5(3) PeEi	4(4)
V_i	13(8) PeVi	11(7) PeVi	7(4) PeVi	6(3) PeVi	5(3) PeVi	5(3) PeVi

Table 2: The number of field and record accesses for basic queries in the partial entity structure (PES).

	$\{R_i\}$	$\{S_i\}$	$\{F_i\}$	$\{L_i\}$	$\{E_i\}$	$\{V_i\}$
R_i	4(3) FuRi	3(2) FuRi	2(1) FuRi	2(1) LuRi	2(1) EuRi	3(2)EuRi
S_i	1(1)	3(2) FuSi	2(1) FuSi	2(1) LuSi	2(1) EuSi	3(2) EuSi
F_i	6(5)	4(3)	10(8) PeFi	2(1) LFi	2(1) PeFi	3(2) PeFi
L_i	8(6)	6(4)	3(3)	9(7) PeLi	2(1) PeLi	3(2) PeLi
E_i	8(4) PeEi	7(3) PeEi	7(3) PeEi	6(2) PeEi	8(4) PeEi	6(5)
V_i	8(5) PeVi	7(4) PeVi	7(4) PeVi	6(3) PeVi	4(2) PeVi	6(4) PeVi

Table 3: The number of field and record accesses for basic queries in the radial edge structure (RES).

7. ANALYSIS OF STORAGE COMPLEXITY

In this section, the storage cost of our data structure is estimated and compared with existing structures. To facilitate the comparison, the following assumptions are made:

- The field for storing a pointer is four bytes in length.
- One byte is the minimum storage unit. Hence, for example, if there are multiple flags in a class and the sum of their storage sizes is less than one byte, only one byte is allocated for them.
- The fields for attributes or geometric data are not counted. Only the storage for topology data is estimated.
- All the linked lists in the data structures are assumed to be singly linked lists except for the list of p-edges in the PES and the list of edge-uses in the RES. Unlike other entities, doubly linked lists are essential for p-edges and edge-uses, because search for the next and previous siblings in the loop and radial cycles needs to be performed frequently.

In order to compare the storage requirements of data structures, the statistical data about the average numbers of topological entities for general non-manifold models are required. Unfortunately, such data is not available for non-manifold models yet, while the data for solid models has been investigated by Wilson [34]. Thus, we estimate and compare the storage costs of data structures using the statistical data for solid models. According to Wilson [34], a solid object usually has only one shell. The average numbers of topological entities for solid models are expressed by the function of the number of faces, *f*. The numbers of loops, edges, and vertices are approximately *f*, *3f*, and *2f*. Each loop contains about six edges, and approximately three edges meet at each vertex. The overall storage size of a data structure is calculated by multiplying the number of each entity by its record size and then summing them up.

The storage costs of several representative solid and non-manifold data structures are presented in Table 4, where the storage cost of our structure is used as a reference for comparison. Comparing to solid data structures such as the winged-edge structure and the half-edge structure, the PES has almost twice as much as storage cost. However, this is natural because the PES has not only new entities such as p-faces and p-vertices but also radial cyclic links in p-edges in order to represent non-manifold conditions with a single unified framework. Note that the data structure of ACIS 5.0 is more compact than the PES for representing solids. However, ACIS data structure is not a complete non-manifold boundary representation. It was initially a kind of half-edge structure for solid modeling, and then it has been extended to represent non-manifold conditions afterwards. For instance, if more than one two-manifold surfaces are connected to a vertex, the edge pointer in the vertex class is set to null and multiple edge pointers are stored in the attribute *vertedge*. Consequently, the ACIS data structure is still inefficient in answering some topological queries even though it was extended to represent non-manifold conditions. For example, if one asks the system to find two shells adjacent to a face, the query function has to traverse all of the shells and their faces throughout the body, because the face class of ACIS includes only one shell pointer.

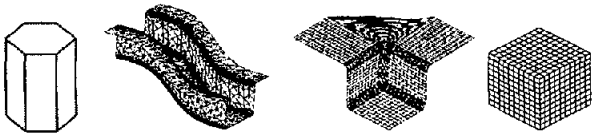
Representation Domain	*Data Structure*	*Wilson's Statistical Data*	
		Storage Size (bytes)	*Ratio to PES*
Non-Manifold	*Radial Edge Structure*	*616f + 71*	*100 %*
	Partial Entity Structure	*303f + 68*	*49 %*
Quasi-Non-Manifold	*ACIS 5.0 Structure*	*195f + 40*	*32 %*
Solid	*Half-Edge Structure*	*176f + 20*	*29 %*
	Winged-Edge Structure	*140f + 20*	*23 %*

Table 4: Storage costs of representative data structures for solid and non-manifold modeling.

In addition to the statistical data, we select several typical solid, sheet, and wireframe models shown in Figure 10 to compare the storage sizes of the PES and the RES with each other. As illustrated in Table 5, our representation requires only about 50% storage size of the RES for solid, sheet, and cellular models, and about 60% for wireframe models. Since a non-manifold model is a combination of solids, sheets, and wireframes, the storage cost of the PES are expected to still hold about 50% that of the RES for general non-manifold models. If we use singly linked lists even for p-edges and edge-uses, the storage size of the PES for solid models is dropped to about 40% that of the RES.

Type	Object	Number of Entities											Total Size (bytes)		Ratio (PES /RES)
		R	S	F (L)	E	V	FU (LU)	EU	VU	PF	PE	PV	RES	PES	
Solid	Statistical Data	2	3	f	3f	2f	2f	12f	12f	2f	6f	2f	616f+71	303f+68	49%
	n-sided Prism	2	3	n+2	3n	2n	2(n+2)	12n	12n	2(n+2)	6n	2n	616n+255	303n+178	49%
Sheet	S-rail Mesh	1	2	3,763	5,697	1,935	7,526	22,578	22,788	7,526	11,289	1,935	1,327,776	639,518	48%
	1/4 Die Mesh	1	2	3,190	4,680	1,671	6,380	19,140	19,440	6,380	9,570	1,671	1,127,430	543,092	48%
Wire	Statistical Data	1	2	0	3f	2f	0	6f	6f	2f	0	6f	272f+45	178f+44	65%
	n-sided Prism	1	2	0	3n	2n	0	6n	6n	3n	0	6n	242n+46	178n+44	65%
	S-rail Mesh	1	2	0	5,697	1,935	0	11,394	11,394	5,697	0	11,394	509,122	328,751	65%
Cell	10x10x10 cubes	1,001	1,003	3,300	3,630	1,331	6,600	26,400	26,400	6,600	13,200	1,331	1,457,303	644,192	44%

Table 5: Storage requirements for some selected models represented in the radial edge structure and the partial entity structure.



(a) (b) (c) (d)

Figure 10: Typical geometric models for storage comparison. (a) an *n*-sided prism (*n*=6). (b) a mesh model of s-rail shape. (c) a mesh model of a 1/4 drawing die. (d) a cellular model with 10x10x10 cubic cells.

8. NON-MANIFOLD EULER OPERATORS

The extended Euler-Poincare formula for *3-D* non-manifold models accepted in our system is as follows.

$$V - E + F - L = S - C + R, \qquad (1)$$

where V, E, F, L, S, C, and R are the numbers of vertices, edges, faces, hole loops, void shells, non-manifold cycles, and regions, respectively [36]. The non-manifold cycle can be interpreted as an independent cycle that is not converted to a face in a wireframe composed of vertices and edges, or a circle that can not be retracted to a point on a surface.

Since there are six independent variables in Eq. (1), at least six independent Euler operators and their six inverse operators are required to manipulate the topological structure of non-manifold models. In this paper, however, in order to facilitate the implementation of high-level modeling operations, four pairs of Euler operators are supplemented to six pairs of basic operators. Furthermore, a pair of non-Euler operators are added for initially creating a model and its infinite region, and finally deleting them. In Table 6, all of the primitive topological operators in our system are listed with the specification of input and output arguments of the functions. Their functionalities are illustrated briefly in Figure 11. According to the naming convention for Euler operators, the following abbreviations are used to represent each operation and topological entity: M (make), K (kill), S (split), J (join), V (vertex), E (edge), F (face), L (hole loop), S (void shell), C (non-manifold cycle), and R (region).

NMT Operators	V	E	F	L	S	C	R
Basic NMT Operators							
MMR (**new_m, new_r**)							
KMR (*new_m*)							
Basic Euler Operators							
MVS (*r*, **new_v, new_s**, *p*)	1	0	0	0	1	0	0
KVS (*new_s*)	-1	0	0	0	-1	0	0
MEV (*v, parent*, **new_e, new_v**, *cv*)	1	1	0	0	0	0	0
KEV (*new_e, new_v*)	-1	-1	0	0	0	0	0
MEC (*s, v1, v2*, **new_e**, *cv*)	0	1	0	0	0	1	0
KEC (*new_e*)	0	-1	0	0	0	-1	0
MFKC (*s, e_list*, **new_f**, *sf*)	0	0	1	0	0	-1	0
KFMC (*new_f*)	0	0	-1	0	0	1	0
MFR (*s, e_list*, **new_f, new_r**, *sf*)	0	0	1	0	0	0	1
KFR (*new_f, new_r*)	0	0	-1	0	0	0	-1
MVL (*f*, **new_v, new_l**, *p*)	1	0	0	1	0	0	0
KVL (*new_l*)	-1	0	0	-1	0	0	0
Supplementary Euler Operators							
SEMV (*e*, **new_e, new_v**, *p*)	1	1	0	0	0	0	0
JEKV (*new_e, new_v*)	-1	-1	0	0	0	0	0
MEF (*l, v1, v2*, **new_e, new_f**, *cv*)	0	1	1	0	0	0	0
KEF (*new_e, new_f*)	0	-1	-1	0	0	0	0
KEML (*e*, **new_l**)	0	-1	0	1	0	0	0
MEKL (*l1, l2, v1, v2*, **new_e**, *cv*)	0	1	0	-1	0	0	0
KEMS (*e*, **new_s**)	0	-1	0	0	1	0	0
MEKS (*s1, s2, v1, v2*, **new_e**, *cv*)	0	1	0	0	-1	0	0

Table 6: Euler operators for non-manifold modeling.

As illustrated in Figure 11, MEV can create a manifold edge or a non-manifold wire edge according to the type of *parent*. If the *parent* is a loop, an edge and a vertex are created in the face of the loop. If the *parent* is a shell, they are created in the region of the shell. A single circuit list of edges *e_list* is given to MFKC and MFR as input to serve as the peripheral loop of the new face *new_f*.

Since all of the high-level modeling operations, such as sweeping and Boolean operations, have been implemented using this set of Euler operators in order to build our kernel modeler, it is already verified that this set is sufficient for the development of a non-manifold modeler.

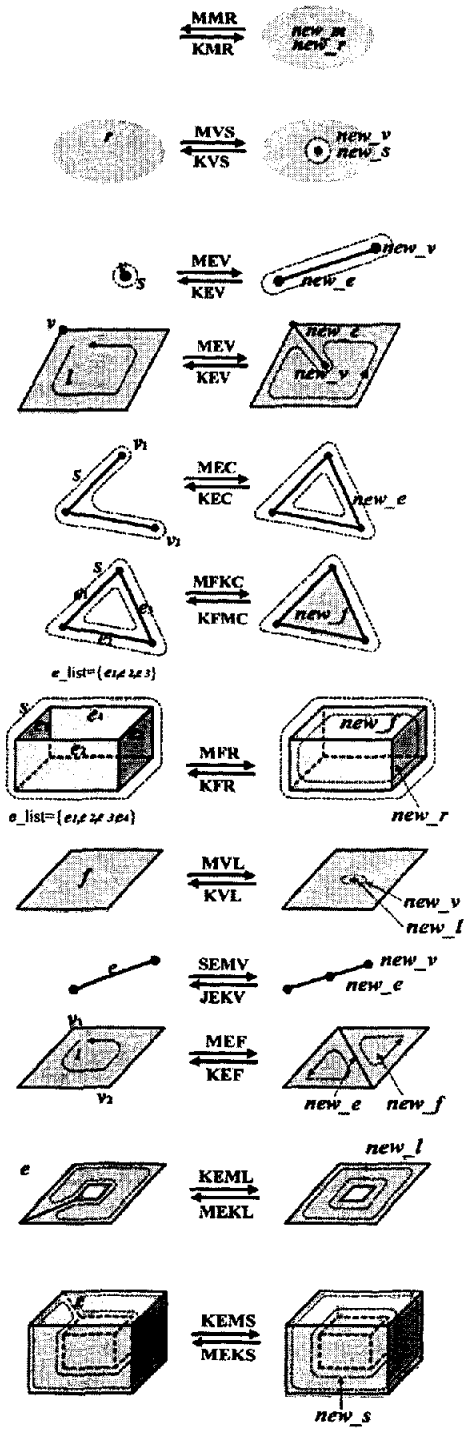


Figure 11: Euler operators for non-manifold modeling.

9. CONCLUSION

In this paper, we propose a compact hierarchical non-manifold boundary representation that allows reducing data storage significantly while maintaining the time efficiency of the radial edge structure, which is the most popular and efficient data structure among the ${}_{16}C_6$ non-manifold B-reps that includes six out of sixteen topological adjacency relationships and the ordering information. The partial topological entities have been introduced to represent the non-manifold conditions at the face, edge, and vertex in an economic and efficient way for the storage and retrieval of topological data.

We showed first that all adjacency relationships between the basic topological entities could be extracted from this data structure using basic query procedures. Next, in order to prove that our representation is as efficient as the RES, we analyzed the time complexities of the basic topological query procedures and compared them with those of the RES. Then, we estimated the storage requirements of our representation for typical solids, wireframes, meshes, and cells, and compared them with those of other existing representations. As a result, it was proven that our representation occupies only half the storage space of the RES.

Our structure practically has a great advantage over the RES. The data structures of ACIS and Parasolid can be converted to our structure very easily, because the coedge of ACIS and the fin of Parasolid are used very similar to the p-edge of our structure. Just by introducing the p-face and the p-vertex, they can be converted to our data structure.

In addition, we propose and implement a set of Euler operators based on the Euler-Poincare formula generalized for three-dimensional non-manifold models. These non-manifold Euler operators guarantee the integrity of non-manifold models, insulate the higher-level of topological functions from the specific data structure, and enable the easy implementation of UNDO functions using the reverse operators, as do the Euler operators for B-rep solid modeling. Since this set was selected through the development of our modeling kernel, it is already verified to be sufficient for the development of a non-manifold modeler that has high-level modeling operations such as sweeping or Boolean operations.

REFERENCES

[1] Baumgart, B., "Winged-edge polyhedron representation," Technical Report CS-320 Stanford Artificial Intelligence Laboratory, Stanford University, CA, USA, 1972

[2] Braid, I., Hillyard, R., and Stroud, I., "Stepwise Construction of Polyhedron in Geometric Modelling", CAD Group Document No.100, University of Cambridge Computer Laboratory, October 1978

[3] Brisson, E., "Representing Geometric Structures in d Dimensions: Topology and Order," Proceedings of the 5th ACM Symposium on Computational Geometry, ACM Press, New York, pp. 218-227, 1989

[4] Cavalcanti, P. R., Carvalho., P. C. P., and Martha, L. F., "Non-manifold modeling: an approach based on spatial subdivision," Computer-Aided Design, Vol. 29, No. 3, pp. 209-220, 1997

[5] Charlesworth, W. W. and Anderson, D. C., "Applications of Non-manifold Topology," Proceedings of the

Computers in Engineering Conference and the Engineering Database Symposium, ASME, pp.103-112, 1995

[6] Choi, Y., "Vertex-based Boundary Representation of Non-manifold Geometric Models," PhD. Thesis, Carnegie Mellon University, August 1989

[7] Crocker, G.A. and Reinke, W. F., "An Editable Non-manifold Boundary Representation," IEEE Computer Graphics & Applications, Vol. 11, No. 2, pp. 39-51, March 1991

[8] Dobkin, D.P. and Laszlo, M.M., "Primitives for the Manipulation of Three-Dimensional Subdivisions," Proceedings of the 3th ACM Symposium on Computational Geometry, ACM Press, New York, pp. 86-99, 1987

[9] Guibas, L. and Stolfi, J., "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," ACM Transactions on Graphics, Vol.4, No.2, pp.74-123, 1985

[10] Gursoz, E.L., Choi, Y., and Prinz, F.B., "Vertex-based Boundary Representation of Non-manifold Boundaries," Wozny, M.J., Turner, J.U., and Preiss, K., editors, Geometric Modeling for Product Engineering, North-Holland, pp. 107-130, 1990

[11] Gursoz, E.L., Choi, Y., and Prinz, F.B., "Boolean Set Operations on Non-Manifold Boundary Representation Objects," Computer-Aided Design, Vol.23, No.1, pp.33-39, 1991

[12] Hansen, H.Ø. and Christensen N.J., "A Model for n-Dimensional Boundary Topology," Proceedings of the 2nd ACM Symposium on Solid Modeling and Applications, Montreal, Canada, pp. 65-73, May 1993

[13] Heisserman, J.A., "A Generalized Euler-Poincare Equation," Proceedings of the 1st ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, pp.533, June 1991

[14] Higashi, M., "High-quality solid-modelling system with free-form surfaces," Computer-Aided Design, Vol. 25, No. 3, pp.172-183, March 1993

[15] Lee, S. H., "Offsetting Operations in Non-manifold Geometric Modeling," Proceedings of the 5th ACM Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, pp. 42-53, June 9-11, 1999

[16] Lienhardt, P., "Subdivision of N-Dimensional Spaces and N-Dimensional Generalized Maps," Proceedings of the 5th ACM Symposium on Computational Geometry, ACM Press, New York, pp. 228-236, 1989

[17] Lienhardt, P., "Topological Models for Boundary Representation: a comparison with n-dimensional generalized maps", Computer-Aided Design, Vol.23, No.1, pp.59–82, January/Febrary 1991

[18] Luo,Y., "Generalized Euler Operators for Non-Manifold Boundary Solid Modeling", Geometric Modelling Studies 1990/3, MTA SZTAKI, Hungary, pp.19-34, 1993

[19] Luo,Y. and L.Gabor, "A Boundary Representation for Form Features and Non-manifold Solid Objects," Proceedings of the 1st ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, June 5-7, 1991, pp.45-60, 1991

[20] Mantyla, M., An Introduction to Solid Modeling, Computer Sceience Press, 1988

[21] Mantyla, M. and Sulonen, R., "GWB: a solid modeler with the Euler operators," IEEE Computer Graphics and Applications, Vol. 2, No. 7, pp. 17-31, September 1982

[22] Marcheix, D. and Gueorguieva, S., "Topological Operators for Non-manifold Modeling," Proceedings of the 30th International Symposium on Automotive Technology and Automation, Mechatronics/Automotive Electronics, Florence, Italy, 16-19 June 1997, pp.173-186, 1997

[23] Masuda, H., "Topological operators and Boolean operations for complex-based nonmanifold geometric models," Computer-Aided Design, Vol. 25, No. 2, pp.119-129, February 1992

[24] Pratt, M. J., "A hybrid feature-based modeling system," In Krause, F. L. and Jansen, H., editors, Advanced Geometric Modeling for Engineering Applications, North-Holland, pp.189-201, 1990

[25] Rossignac, J. and O'Conner, M.A., "SGC: A Dimensional-independent Model for Pointsets with Internal Structures and Incomplete Boundaries", Geometric Modeling for Product Engineering, North-Holland, pp. 145 – 180, 1990

[26] Saxena, M., Finnigan, P.M., Graichen, C.M., Hathaway, A.F., and Parthasarathy, V.N., "Octree-Based Automatic Mesh Generation for Non-Manifold Domains," Engineering with Computers, Vol. 11, pp.1-14, 1995

[27] Shimada, K. and D. C. Gossard, "Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing", Proceedings of the 3rd Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, UT., USA, pp. 409-419, 1995

[28] Solid Modeling Solutions, http://www.smlib.com

[29] Spatial Technology Inc., ACIS 3D Toolkit 5.0, 1999

[30] Sriram, R.D., Wong, A., and He, L.-X., "GNOMES: an object-oriented nonmanifold geometric engine," Computer-Aided Design, Vol. 27, No. 11, pp. 853-868, 1995

[31] Weiler, K "The Radial Edge Structure: a Topological Representation for Non-manifold Geometric Boundary Modeling", Wozny, M.J., McLauhlin,H.W and Encarnçao, J.L., editors, Geometric Modeling for CAD Applications, Noth-Holland, pp.3-36, 1988

[32] Weiler, K "Boundary Graphs Operators for Non-Manifold Geometric Modeling Topology Representations", Wozny, M.J., McLauhlin ,H.W and Encarnçao, J.L., editors, Geometric Modeling for CAD Applications, Noth-Holland, pp.37-66, 1988

[33] Weiler, K., "Generalized sweep operations in the non-manifold environment," Geometric modeling for product engineering, Wozny, M.J., Turner, J.U., and Preiss, K., Edts., Elsevier Sci, North Holland, 1990

[34] Wilson, P.R., "Data Transfer and Solid Modeling", Geometric Modeling for CAD Applications, M.M.Wony, H.W.McLaughlin, and J.L.Encarnacao (Editors.), Elsevier Science Publishers B.V., North-Holland, pp.217 – 254, 1988

[35] Woo, T.C., "A Combinational Analysis of Boundary Data Structure Schemata," IEEE Computer Graphics and Applications, pp.19-27, March 1985

[36] Yamaguchi, Y. and Kimura, F., " Nonmanifold Topology Based on Coupling Entities," IEEE Computer Graphics and Applications, Vol.15, No.1, pp.42-50, January 1995