

Partial Entity Structure: A Compact Boundary Representation for Non-Manifold Geometric Modeling

Sang Hun Lee

Graduate School of Automotive Engineering,
Kookmin University, Korea
e-mail: shlee@kookmin.ac.kr

Kunwoo Lee

Mem. ASME
School of Mechanical and Aerospace
Engineering,
Seoul National University, Korea
e-mail: kunwoo@snu.ac.kr

Non-manifold boundary representations have become very popular in recent years and various representation schemes have been proposed, as they represent a wider range of objects, for various applications, than conventional manifold representations. As these schemes mainly focus on describing sufficient adjacency relationships of topological entities, the models represented in these schemes occupy storage space redundantly, although they are very efficient in answering queries on topological adjacency relationships. To solve this problem, in this paper, we propose a compact as well as fast non-manifold boundary representation, called the partial entity structure. This representation reduces the storage size to half that of the radial edge structure, which is one of the most popular and efficient of existing data structures, while allowing full topological adjacency relationships to be derived without loss of efficiency. 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. [DOI: 10.1115/1.1433486]

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

1 Introduction

As a key element in CAD/CAM applications, geometric modeling systems have evolved to increase their representation domains from wireframes, to surfaces, to solids, and finally to non-manifold objects. Unlike previous geometric modellers, 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 models as follows:

- The non-manifold model 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 [1,2]. For instance, it handles 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 [3,4], offset polyhedral models for tool path generation, and so on.
- Boolean operations are closed in the representation domain of non-manifold models, unlike solid models [5–7]. 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 [5,8,9]. By using this merged set, B-rep models can be reshaped independently of their construction Boolean sequences [9], and a feature-based modeler based on B-rep can be easily implemented [9,10].
- Traditional solid modeling functions such as 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 [11]. In addition, a thin-walled solid model can be generated efficiently by sheet modeling and offsetting [12].

The research concerned with the foundation of non-manifold modeling systems can be categorized into three groups: the design of topological representation schemes, the specification of a set of primitive topological operators, and the implementation of various high-level modeling capabilities 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 be a critical problem, particularly for models in which topological data storage is more dominant than geometric data, such as tessellated or mesh models. For example, a cellular model composed of one million cubical cells requires more than 1 GB of storage only for its topological data if it is represented in the radial edge structure. Therefore, it is desirable to devise a new representation scheme that is more compact, but as efficient as the existing schemes.

To fulfil this requirement, in this paper, we propose a compact as well as fast non-manifold boundary representation, called the *partial entity structure* (PES). This representation allows the reduction of storage to approximately half that of the radial edge structure, while still allowing full topological adjacency relationships to be derived without loss of efficiency. In order to verify this improvement, the time and storage efficiency of the partial entity structure are investigated and compared with those of existing schemes.

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 rep-

Contributed by the Computer Aided Product Development (CAPD) Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received Aug. 2001; revised manuscript received Nov. 2001. Associate Editor: D. Anderson, K. Lee.

resent 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 analyze and compare the time and storage complexity of our structure with those of other existing structures, especially the radial edge structure. Section 8 presents our conclusions.

2 Related Work

The first significant work on the non-manifold B-rep was carried out by Weiler [7]. 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 the topological entities, such as face-use, loop-use, edge-use, and vertex-use, which 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 [13], 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 the ordering of edges and faces about a vertex is a *disk cycle*. Therefore, the RES represents explicitly the radial and loop cycles. 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. This is because the RES does not keep any significant inclusion relationships between the incident two-manifold surfaces to a non-manifold vertex. A vertex adjacent to two or more two-manifold surfaces or wire edges is a typical 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 extracts this information using both 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 entities are introduced to represent the inclusive relationships between the local regions at a vertex [5,14]. 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 geometric data whenever necessary would be more rational from a practical viewpoint, because the zone and disk information is not frequently used in the normal geometric modeling process, and maintaining such information at every topology manipulation incurs a very high cost.

Yamaguchi and Kimura introduced six coupling entities to represent the neighbourhoods and boundaries of basic topological entities, and suggested a data structure based on the coupling entities [13]. 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 appears 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 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 [15]. The cells are open connected subsets of n -dimensional manifolds that 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 [16].

Lee and Lee suggested that the half-edge data structure for solid modelling can be extended to accommodate non-manifold geometric models in storage efficient way by introducing auxiliary topological entities like partial entities [17]. However, they did not show how much it is efficient in inquiring topological information and how much it is efficient in storing typical non-manifold models. They dealt with these issues on their recent conference publication [18], in which the concept of the partial entities is clearly defined and explained in more detail with comparison of other similar topological entities. In addition, the methods to measure the time and storage efficiency of non-manifold data structure are suggested and applied to the existing structures as well as the PES to verify the storage and time efficiency of the PES. This paper is an extended and revised version of Reference [18].

On the other hand, several works have defined topological representations for subdivided n -manifolds. Brisson defines the 'cell-tuple structure' to represent the incidence and ordering information in a subdivided n -manifold [19]. Lienhardt defines the 'n-G-map' and the 'n-map', based on combinatorial maps [16,20]. Hansen and Christensen proposed a hierarchical 'split-element' representation called the hyper-data structure [21]. These data structures include the 'quad-edge' data structure of Guibas and Stolfi [22] and the 'facet-edge' data structure of Dobkin and Laszlo [23] as special cases in dimensions 2 and 3, respectively.

3 Design Strategy for Optimal Data Structure

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 [24]. The optimal data structure for a general or specific set of applications can be constructed on the basis of his 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' as illustrated in Fig. 4 of Reference [24]. 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, they should investigate the frequency of topological queries first, and then store the adjacency relations in order of frequency. By applying these measures, it was discovered that the winged edge data structure [25] is very close to the globally optimal data structure. Furthermore, Woo claimed that a new data structure that has a lower storage requirement 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 $F \rightarrow E$, $E \rightarrow V$, $V \rightarrow E$, and $E \rightarrow F$ 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 $S \rightarrow F$, $F \rightarrow S$, $F \rightarrow L$, $L \rightarrow F$, $L \rightarrow E$, $E \rightarrow L$, $E \rightarrow V$, and $V \rightarrow E$ 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.

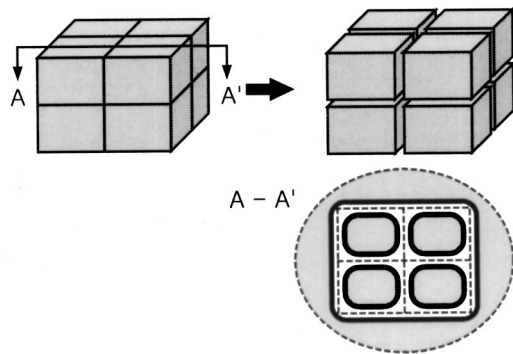


Fig. 1 Regions and their boundaries in a cellular model represented in the radial edge structure (RES)

To date we are unaware of any reported 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 modellers such as SMLib of Solid Modelling Solutions [26], but also academic modellers researching various non-manifold applications have adopted the RES as their topological framework [3,27]. However, it is also true that there has been little analytic rationalization of its time and storage efficiency by its users.

In non-manifold models, there are six basic topological entities: R, S, F, L, E, and V, and thus, 36 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 16: $R \rightarrow F$, $F \rightarrow R$, $F \rightarrow E$, $E \rightarrow F$, $E \rightarrow V$, and $V \rightarrow E$. If a region is not bounded by any wireframe, the RES describes the boundaries of a region, just as the symmetric data structure describes the boundaries of a solid. Thus, we can conjecture that the RES is the fastest data structure in the ${}_{16}C_6$ storage class to which it belongs, 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 cannot be said to be optimal. As illustrated in Fig. 1, in the RES, each region of a non-manifold model has its complete description of its boundaries just like a solid B-rep model. As a result, the boundaries of regions are stored twice in different orientations. Therefore, if we reduce the redundancy of the RES 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 is expensive to maintain this information at every primitive topological operation while this information is rarely used in actual geometric modelling 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. In addition, the ordering information of the RES is also maintained by our new representation.

To achieve this goal, we made a new idea that our new representation focuses on the frame of the model, not volumes, unlike the RES, as illustrated in Fig. 2. Here, if the boundary information of a region is necessary, it is derived from the faces of the frame considering the direction to the region. Therefore, we can expect

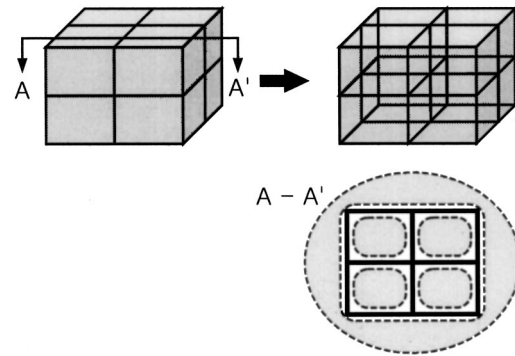


Fig. 2 The basic idea of our representation: representing the frame of a model rather than its volumes

that its storage size will be reduced to about half that of RES. However, a not a little drop in time efficiency may not be avoided due to the trade-off relationship between the time and the storage efficiency of a data structure.

In this paper, 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 [9,23,28]. 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 an 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 wish 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 identical to 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). Together, they are called *partial topological entities* or *partial entities*. 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 Fig. 3(a). The p -edge is introduced for the non-manifold condition where more than two faces are connected to an edge as illustrated in Fig. 3(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 Fig. 3(c). Now we will explain the partial entities along with the primary entities in the following sections.

4.1 Partial Faces. In our representation, a model is the highest level of topological entity that can be an object of manipulation in the non-manifold modeller. A model is composed of one or more regions: one infinite open region and zero or more finite closed regions. For example, the box with an inner partition shown in Fig. 3(a) is represented by a non-manifold model with

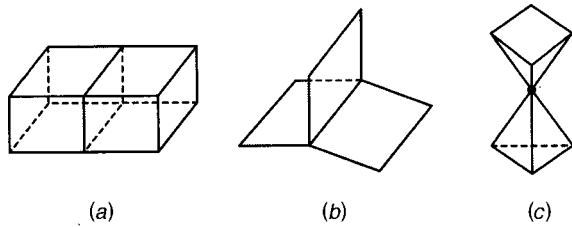


Fig. 3 Typical non-manifold conditions (a) a face with two incident regions; (b) an edge with three incident faces; (c) a vertex with two incident two-manifold surfaces

three regions: in infinite external region R_0 and two closed internal regions R_1 and R_2 as illustrated in Fig. 4. 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, known as the shells. A region has a single peripheral shell and zero or more void shells. In Fig. 4, 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 Fig. 4. However, if a single vertex or wireframe edges are associated with the shell, those portions are not oriented exceptionally.

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. Then, 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 [29]. As shown in Fig. 5(a), 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 Figs. 4 and 5. However, there are exceptions when an isolated vertex or wireframe edges are included in a

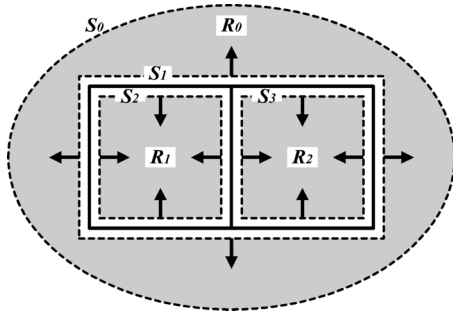


Fig. 4 Examples of regions and shells in a non-manifold model

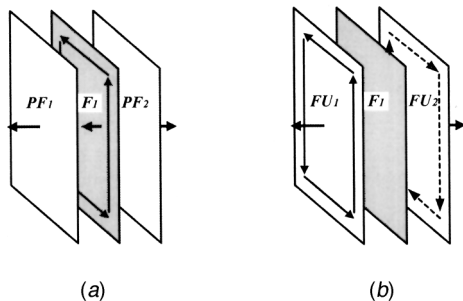


Fig. 5 Example of partial faces and face-uses: (a) two partial faces for a face; (b) two face-uses for a face

shell. In that case, an unoriented p-face is included for the consistency of the data structure. The C++ implementation of such exceptional cases is illustrated in more detail in Section 5.

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. As illustrated in Fig. 5, the face-use has its own boundaries in the form of loop-uses, whereas the p-face does not include any boundaries. The faces of the RES, the VBR, and the CES have no data for their boundaries, whereas the face of our representation has its own boundary data. Because the boundaries of a face-use are coincident with those of its mating face-use, except the orientation, one of them is redundant. This argument is also effective for 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 total storage usage of our representation becomes slightly less than half 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, whereas 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 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, two shells for the faces and the wire edges should represent such a region boundary. 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 zero 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, whereas the hole loop is the inner boundary. The peripheral loop has a counter-clockwise direction with respect to the normal of the face geometry whereas 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 that 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, whereas it always neighbors two faces in a two-manifold model. Since an edge should serve as the boundary of the incident faces, we split an edge into as many p-edges as incident faces. For instance, the p-edges in the non-manifold sheet model shown in Fig. 3(b) are illustrated in Fig. 6(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 Fig. 6(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. This special case is illustrated with a schematic diagram in Section 5.

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 Fig. 6(b), the p-edges around an edge are ordered following the right-hand rule with

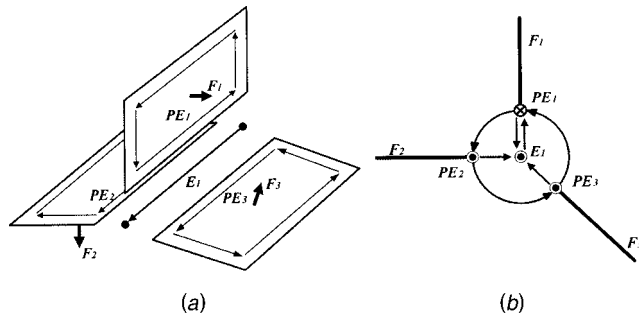


Fig. 6 Partial edges in loops and edges: (a) partial edges ordered in the corresponding loop; (b) partial edges ordered in the corresponding edge

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 divided in two. The C++ implementation of the class for the p-edge is described in Section 5.

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, whereas 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 explained 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 [30], 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, whereas 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, whereas 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. 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 repre-

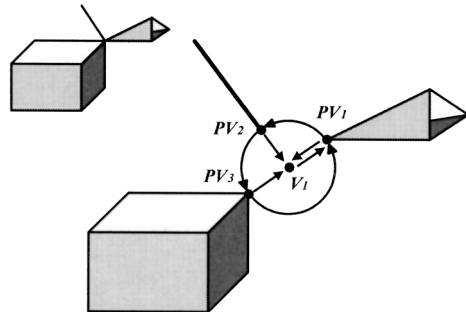


Fig. 7 Example of partial vertices around a vertex

sentation. The discussion about Parasolid can be covered 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. 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 Fig. 7, 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-vertex.

In the C++ implementation of our data structure, 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 in the schematic diagram of our representation in Section 5.

Since a p-vertex points to only one edge of the connected edges, the algorithms for searching for various adjacent topological entities need to be developed. In Section 6, 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.

5 Data Structure

The hierarchical data structure to store these topological entities with their relationships is illustrated in Fig. 8. The data structure is composed of the topological and geometrical parts similar to other traditional B-reps. This data structure is named the *Partial Entity Structure* (PES) in this paper. In Fig. 8, each arrow represents a pointer to another entity. The solid arrows are for the normal

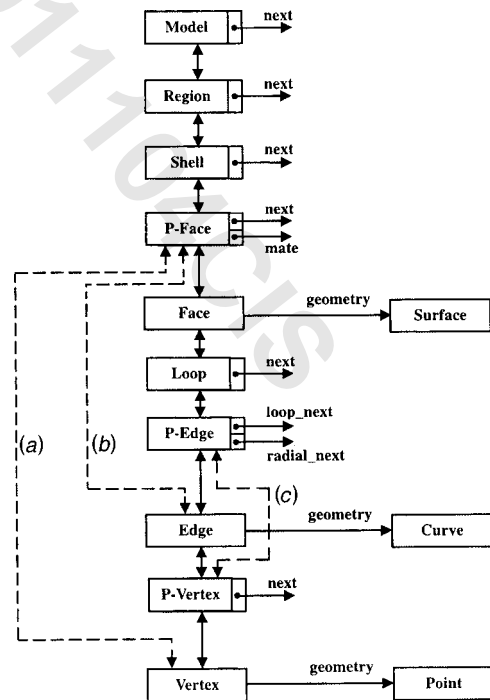


Fig. 8 Schematic diagram of the partial entity structure

```

class Entity {
    int _id;
    Attribute *_attribute;
};
class Model : public Entity {
    Model *_next; // next model
    Region *_region; // list of regions
    Face *_face; // list of faces
    Edge *_edge; // list of edges
    Vertex *_vertex; // list of vertices
};
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 {
    Face *_next; // link field of the face list of a model
    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 {
    Edge *_next; // link field of the edge list of a model
    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; // next p-vertex associated with _vertex
    Entity *_parent; // parent entity: an edge or a p-edge
    Vertex *_vertex; // mother vertex
};
class Vertex : public Entity {
    Vertex *_next; // link field of the vertex list of a model
    Entity *_parent; // parent entity: a p-vertex or a p-face
    Point *_geomerty; // position
};

```

Fig. 9 Implementation of the partial entity structure with the classes in C++

cases, while the dotted lines (a), (b), and (c) in Fig. 8 indicate three exceptional cases, respectively: (a) single-vertex shells, (b) wire edges, and (c) single-vertex loops. 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 a wire edge represents an edge that does not have any incident face.

The implementation of our data structure with the C++ classes is shown in Fig. 9. Each parent node points 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. For p-edges, it is frequently required to find the previous, or next, p-edge from the current one in the loop or radial cycle. To respond to this query efficiently, 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.

In the case of single-vertex shells, a shell has only one p-face. The *_child* field of the p-face contains a pointer to an isolated vertex, and its *_orient* field is set as *unoriented*. The *_parent* field of the isolated vertex contains a pointer to the p-face. In the case of wire edges, each wire edge is wrapped with a p-face. The *_child* field of the p-face points to the corresponding wire edge, and its *_orient* field is set as *unoriented*. The *_parent* field of the wire edge points to the corresponding p-face. In the case of single-vertex loops, a loop has only one p-edge. The *_child* field of the p-edge contains a pointer to a p-vertex for the isolated vertex, and its *_orient* field is set as *unoriented*. The *_parent* field of the p-vertex contains a pointer to the p-edge, and its *_vertex* field points to the isolated vertex.

6 Analysis of Time Complexity

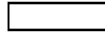

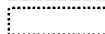
The time complexity of a data structure is usually evaluated by measuring the response time of each basic query that returns topological entities of a specific type that are adjacent to a given entity. In this paper, we also adopted this method to compare the time efficiency of our structure with that of the RES. Let R denote the region, S the shell, F the face, L the loop, E the edge, V the vertex, Pf the p-face, Pe the p-edge, Pv the p-vertex, Fu the face-use, Lu the loop-use, Eu the edge-use, and Vu the vertex-use, respectively. If A and B represent one of the entities listed above, we 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
$\langle A_i \rangle$ or $\langle A_i \rangle^m$	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

The basic topological entities used in existing non-manifold boundary representations are of six types: region, shell, face, loop, edge, and vertex. Hence, there are 36 possible adjacency relationships between two types of entities, as illustrated in the adjacency relationship matrix in Table 1. This matrix also briefly describes the algorithms of query procedures that collect topological entities of a specific type that are adjacent to a given entity. This matrix verifies that our representation can also provide the same adjacency relationships, including the ordering information, as does the RES. 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, whereas 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 26 adjacency relationships are derived from the existing adjacency relationship information. Since all 36 adjacency relationships for a non-manifold

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

Reference Entity	Adjacent Group Entity					
	Regions	Shells	Faces	Loops	Edges	Vertices
Region	$\{\{R_i\}^{FS_i SR_i}\}$	$\{S_i\}^{SR_i}$	$\{\{F_i\}^{FS_i SR_i}\}$	$\{\{L_i\}^{LF_i FS_i SR_i}\}$	$\{\{<E_i>^{EL_i LF_i FS_i SR_i}\}\}$	$\{\{<V_i>^{EL_i LF_i FS_i SR_i}\}\}$
Shell	$\{R_i\}^1$	$\{S_i\}^{FS_i}$	$\{F_i\}^{FS_i}$	$\{\{L_i\}^{LF_i FS_i}\}$	$\{\{<E_i>^{EL_i LF_i FS_i}\}\}$	$\{\{<V_i>^{EL_i LF_i FS_i}\}\}$
Face	$\{R_i\}^2$	$\{S_i\}^2$	$\{\{<F_i>^2>^{EL_i LF_i}\}\}$	$\{L_i\}^{LF_i}$	$\{<E_i>^{EL_i LF_i}\}$	$\{<V_i>^{EL_i LF_i}\}$
Loop	$\{R_i\}^2$	$\{S_i\}^2$	$\{F_i\}^1$	$\{<L_i>^2>^{EL_i}\}$	$\{<E_i>^{EL_i}\}$	$\{<V_i>^{EL_i}\}$
Edge	$\{<R_i>^{LE_i}\}$	$\{<S_i>^{LE_i}\}$	$\{<F_i>^{LE_i}\}$	$\{<L_i>^{LE_i}\}$	$\{\{<E_i>^2>^{LE_i}\}\}$	$\{<V_i>^2\}$
Vertex	$\{R_i\}^{EV_i}$	$\{S_i\}^{EV_i}$	$\{F_i\}^{EV_i}$	$\{L_i\}^{EV_i}$	$\{E_i\}^{EV_i}$	$\{V_i\}^{EV_i}$

 A full adjacency relationship is stored (in normal conditions)
 A fractional relationship is stored (in normal conditions)
 A full adjacency relationship is stored (in singular conditions)

model can be derived from our representation, it can be said that our representation is complete, according to Weiler's definition [7].

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 access and at least requires a main memory access. In most of the 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 show the calculation of 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 10 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. As in 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 Fig. 10, the *_next* field of a shell is accessed once in line 1, and the *_next*,

```

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;
    
```

Fig. 10 Query procedure for finding faces adjacent to a given region

_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 in the above formula, the dominant term of the 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 use FuR_i instead of PfR_i .

In the same manner, we can count the total number of record accesses. The record of each shell is accessed once in line 1, and the p-face records in a shell are accessed as many times as the number of p-faces in the shell in lines 2. Therefore, the total number of record accesses is $SR_i + FuR_i$, and its dominant term is FuR_i .

As 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 Fig. 11. Note that in this algorithm, each of the edges adjacent to a vertex is visited only once, even though 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.

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*; 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 the 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

```

EDGES-OF-VERTEX ( $v, e\_list$ )
1  for each p-vertex  $pv$  at  $v$  do
2  EDGES-OF-PVERTEX ( $pv, pv \rightarrow\_edge, e\_list$ );
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 =  $pe \rightarrow\_prev\_in\_loop \rightarrow\_edge$ ;
8  else
9  next_e =  $pe \rightarrow\_next\_in\_loop \rightarrow\_edge$ ;
10 if next_e is unvisited
11 EDGES-OF-PVERTEX ( $pv, next\_e, e\_list$ );
12 }
    
```

Fig. 11 Query procedure for finding edges adjacent to a given vertex.

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-PVERTEX, 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 somewhat less than that of the RES. Actually, because the RES stores directly all of the edges adjacent to a vertex using vertexes, it is always faster than our representation in the following queries for vertexes: $V_i \rightarrow \{R_i\}$, $V_i \rightarrow \{S_i\}$, $V_i \rightarrow \{F_i\}$, $V_i \rightarrow \{L_i\}$, $V_i \rightarrow \{E_i\}$, 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 carried out for the RES and its result is summarized as a matrix in Table 3. The figure outside the parentheses in each box denotes the number of field accesses in a query, and the figure inside the parentheses denotes the number of record accesses. To facilitate the comparison, we selected more convenient counting variables between the RESs and the PESs based on such special relations 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 five queries. As a result, it is proven that the PES has almost the same time efficiency as the 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:

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	5(3) FuR_i	4(2) FuR_i	3(1) FuR_i	1(1) LuR_i	3(1) EuR_i	4(2) EuR_i
S_i	1(1)	4(2) FuS_i	3(1) FuS_i	1(1) LuS_i	3(1) EuS_i	4(2) EuS_i
F_i	6(5)	4(3)	8(5) PeF_i	1(1) LF_i	3(1) PeF_i	3(2) PeF_i
L_i	7(6)	5(4)	1(1)	6(3) PeL_i	3(1) PeL_i	3(2) PeL_i
E_i	8(6) PeE_i	7(5) PeE_i	3(2) PeE_i	2(1) PeE_i	5(3) PeE_i	4(4)
V_i	13(8) PeV_i	11(7) PeV_i	7(4) PeV_i	6(3) PeV_i	5(3) PeV_i	5(3) PeV_i

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

	$\{R_i\}$	$\{S_i\}$	$\{F_i\}$	$\{L_i\}$	$\{E_i\}$	$\{V_i\}$
R_i	4(3) FuR_i	3(2) FuR_i	2(1) FuR_i	2(1) LuR_i	2(1) EuR_i	3(2) EuR_i
S_i	1(1)	3(2) FuS_i	2(1) FuS_i	2(1) LuS_i	2(1) EuS_i	3(2) EuS_i
F_i	6(5)	4(3)	10(8) PeF_i	2(1) LF_i	2(1) PeF_i	3(2) PeF_i
L_i	8(6)	6(4)	3(3)	9(7) PeL_i	2(1) PeL_i	3(2) PeL_i
E_i	8(4) PeE_i	7(3) PeE_i	7(3) PeE_i	6(2) PeE_i	8(4) PeE_i	6(5)
V_i	8(5) PeV_i	7(4) PeV_i	7(4) PeV_i	6(3) PeV_i	4(2) PeV_i	6(4) PeV_i

- 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, one byte is allocated for them.
- The fields for attributes or geometric data are not counted. Only the storage for topology data is estimated.
- Data structures are compared in their original forms without any modification. Although a data structure has some redundant parts, we leave it as it is when counting its storage cost. If the implementation of a data structure was not published, we assume that it uses singly linked lists for storing a set of adjacent topological entities.

In order to compare the storage requirements of data structures, the statistical data for 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 [31]. Thus, we estimate and compare the storage costs of data structures using the statistical data for solid models. According to Wilson [31], 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.

The storage costs of several representative solid and non-manifold data structures are presented in Table 4, where the stor-

Table 4 Storage costs of representative data structures for solid or non-manifold modelling calculated using Wilson's statistical data for solid models

Representation Domain	Data Structure	Storage Size (bytes)	Ratio to PES
Non-Manifold	Vertex-Based Representation	$858f + 87$	126 %
	Radial Edge Structure	$680f + 95$	100 %
	Partial Entity Structure	$303f + 68$	45 %
	Selective Geometric Complex	$469f + 12$	69 %
Quasi-Non-Manifold	ACIS 5.0 Structure	$195f + 40$	29 %
Solid	Half-Edge Structure	$176f + 20$	26 %
	Winged-Edge Structure	$140f + 20$	23 %

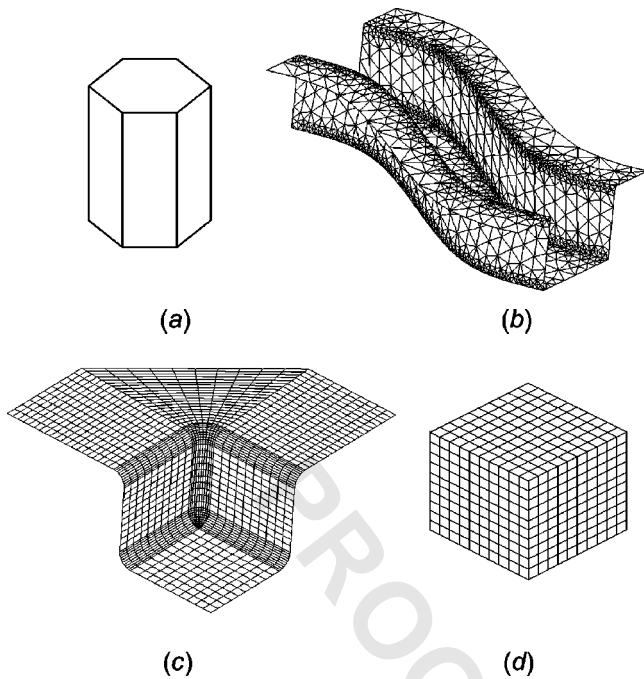


Fig. 12 Typical geometric models for storage comparison: (a) an n -sided prism ($n=6$); (b) a mesh model of s-rail shape; (c) a mesh model of a 1/4 drawing die; (d) a cellular model with $10 \times 10 \times 10$ cubic cells

age cost of the RES is used as a reference for comparison. According to the assumptions mentioned above, the VBR and the SGC are assumed to use singly linked lists as we are unaware of any reported research work on their implementation to date. As shown in Table 4, the PES is the most storage-efficient one among the representative boundary representations listed in the table. As compared to solid data structures such as the winged edge structure and the half edge structure, the PES has almost twice the storage cost. However, this is natural because the PES has not only new entities, such as p-faces and p-vertices, but also radial link fields 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, the ACIS data structure is not a complete non-manifold boundary representation. It was initially a half edge structure for solid modelling, and it has since been extended to represent non-manifold conditions. 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 *verledge*. 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

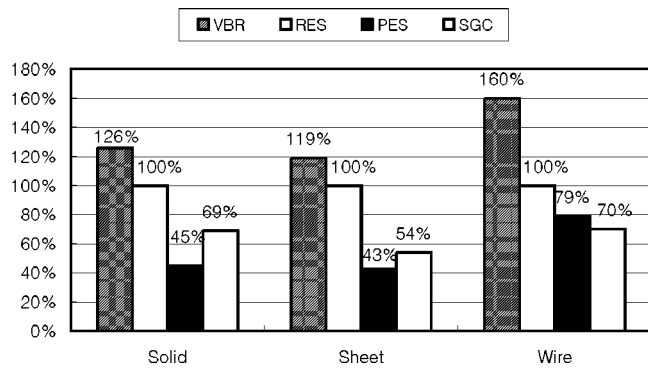


Fig. 13 Storage costs of representative non-manifold data structures for the selected models in Table 5

query function must traverse all of the shells and their faces throughout the body, because the face class of ACIS includes only one shell pointer.

In addition to the statistical data, we select several typical solid, sheet, and wireframe models, as shown in Fig. 12, to compare the storage sizes of the PES and the RES. As illustrated in Table 5, our representation requires only approximately 40% of the storage of the RES for solid, sheet, and cellular models, and about 80% for wireframe models. Since a non-manifold model is a combination of solids, sheets, and wireframes, the storage costs of the PES are expected to still be about 50% that of the RES for general non-manifold models. In addition, we also investigated the storage costs of several representative non-manifold data structures for the selected models in Table 5, and the result is summarized as graphs in Fig. 13.

8 Conclusion

In this paper, we propose a compact hierarchical non-manifold boundary representation that allows significantly reduced data storage 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. It includes six out of 16 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

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

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	680f+95	303f+68	45%
	n-sided Prism	2	3	n+2	3n	2n	2(n+2)	12n	12n	2(n+2)	6n	2n	680n+311	303n+176	45%
Sheet	S-rail Mesh	1	2	3,763	5,697	1,935	7,526	22,578	22,788	7,526	11,289	1,935	1,479,152	639,518	43%
	1/4 Die Mesh	1	2	3,190	4,680	1,671	6,380	19,140	19,440	6,380	9,570	1,671	1,256,246	543,092	43%
Wire	Statistical Data	1	2	0	3f	2f	0	6f	6f	2f	0	6f	224f+62	178f+44	79%
	S-rail Mesh	1	2	0	5,697	1,935	0	11,394	11,394	5,697	0	11,394	417,986	328,751	79%
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	1,623,723	644,192	40%

RES. From a practical viewpoint, our structure has a great advantage over the RES. The data structures of ACIS and Parasolid can be converted to our structure very easily, because the co-edge of ACIS and the fin of Parasolid are very similar to the p-edge of our structure. Only by introducing the p-face and the p-vertex, they can be converted to our structure.

Based on the partial entity structure, we have also developed an object-oriented non-manifold geometric modeling kernel system with open architecture, called NGM. All of the topological or geometric information of non-manifold models can be accessed directly and freely through the C++ class libraries, and all of the modelling and interrogation functions are provided in the form of APIs as well as C++ class libraries. This kernel provides not only high-level modeling operations like Boolean and sweeping operations, but also a set of Euler operators based on the generalized Euler-Poincare formula for 3-D non-manifold models [18]. Through the development of a non-manifold modeler, it was confirmed that our data structure is sound and easy to be manipulated. Further work is under way to bring this kernel to completion by adding more modelling capabilities and to apply it to various application areas.

Acknowledgments

This work was partially supported by Postdoctoral Fellowship Program from Korea Science and Engineering Foundation. The first author is grateful to Prof. Tony C. Woo for his stimulating this research by giving instruction in analysis of data structures during the first author's sabbatical year visit.

References

- [1] Charlesworth, W. W., and Anderson, D. C., 1995, "Applications of Non-manifold Topology," Proceedings of International Computers in Engineering Conference and the ASME Engineering Database Symposium, Boston, MA, pp. 103–112.
- [2] Sriram, R. D., Wong, A., and He, L.-X., 1995, "GNOMES: An Object-oriented Nonmanifold Geometric Engine," *Comput.-Aided Des.*, **27**, No. 11, pp. 853–868.
- [3] Saxena, M., Finnigan, P. M., Graichen, C. M., Hathaway, A. F., and Parthasarathy, V. N., 1995, "Octree-Based Automatic Mesh Generation for Non-Manifold Domains," *Eng. Comput.*, **11**, pp. 1–14.
- [4] Shimada, K., and Gossard, D. C., 1995, "Bubble Mesh: Automated Triangular Meshing of Non-manifold Geometry by Sphere Packing," Proceedings of the 3rd Symposium on Solid Modeling and Applications, Salt Lake City, UT, pp. 409–419.
- [5] Gursoz, E. L., Choi, Y., and Prinz, F. B., 1990, "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.
- [6] Gursoz, E. L., Choi, Y., and Prinz, F. B., 1991, "Boolean Set Operations on Non-Manifold Boundary Representation Objects," *Comput.-Aided Des.*, **23**, No. 1, pp. 33–39.
- [7] Weiler, K., 1988, "The Radial Edge Structure: a Topological Representation for Non-manifold Geometric Boundary Modeling," M. J. Wozny, H. W. McLaughlin, and J. L. Encarnação, editors, *Geometric Modeling for CAD Applications*, North-Holland, pp. 3–36.
- [8] Crocker, G. A., and Reinke, W. F., 1991, "An Editable Non-manifold Boundary Representation," *IEEE Comput. Graphics Appl.*, **11**, No. 2, pp. 39–51, March.
- [9] Masuda, H., 1992, "Topological Operators and Boolean Operations for Complex-based Nonmanifold Geometric Models," *Comput.-Aided Des.*, **25**, No. 2, pp. 119–129.
- [10] Pratt, M. J., 1990, "A Hybrid Feature-based Modeling System," in *Advanced Geometric Modeling for Engineering Applications*, F. L. Krause and H. Jansen, editors, North-Holland, pp. 189–201.
- [11] Weiler, K., 1990, "Generalized Sweep Operations in the Non-manifold Environment," *Geometric Modeling for Product Engineering*, M. J. Wozny, J. U. Turner, and K. Preiss, editors, Elsevier Science North-Holland.
- [12] Lee, S. H., 1999, "Offsetting Operations in Non-manifold Geometric Modeling," Proceedings of the 5th ACM Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, pp. 42–53.
- [13] Yamaguchi, Y., and Kimura, F., 1995, "Nonmanifold Topology Based on Coupling Entities," *IEEE Comput. Graphics Appl.*, **15**, No. 1, pp. 42–50.
- [14] Choi, Y., 1989, "Vertex-based Boundary Representation of Non-manifold Geometric Models," Ph.D. Thesis, Carnegie Mellon University.
- [15] Rossignac, J., and O'Conner, M. A., 1990, "SGC: A Dimensional-independent Model for Pointsets with Internal Structures and Incomplete Boundaries," *Geometric Modeling for Product Engineering*, North-Holland, pp. 145–180.
- [16] Lienhardt, P., 1991, "Topological Models for Boundary Representation: A Comparison with n-dimensional Generalized Maps," *Comput.-Aided Des.*, **23**, No. 1, pp. 59–82.
- [17] Lee, S. H., and Lee, K., 1996, "Compact Boundary Representation and Generalized Euler Operators for Non-manifold Geometric Modeling," *Transaction of the SCCE*, **1**, No. 1, pp. 1–19 (in Korean).
- [18] Lee, S. H. and Lee, K., 2001, "Partial Entity Structure: A Fast and Compact Non-Manifold Boundary Representation Based on Partial Topological Entities," *The Sixth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, pp. 159–170.
- [19] Brissou, E., 1989, "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.
- [20] 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.
- [21] Hansen, H. Ø. and Christensen N. J., 1993, "A Model for n- Dimensional Boundary Topology," Proceedings of the 2nd ACM Symposium on Solid Modeling and Applications, Montreal, Canada, pp. 65–73.
- [22] Guibas, L., and Stolfi, J., 1985, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," *ACM Trans. Graphics*, **4**, No. 2, pp. 74–123.
- [23] Dobkin, D. P. and Laszlo, M. M., 1987, "Primitives for the Manipulation of Three-Dimensional Subdivisions," Proceedings of the 3rd ACM Symposium on Computational Geometry, ACM Press, New York, pp. 86–99.
- [24] Woo, T. C., 1985, "A Combinational Analysis of Boundary Data Structure Schemata," *IEEE Comput. Graphics Appl.*, **5**, No. 3, pp. 19–27.
- [25] Baumgart, B., 1972, "Winged-edge Polyhedron Representation," Technical Report CS-320 Stanford Artificial Intelligence Laboratory, Stanford University, CA.
- [26] Solid Modeling Solutions, <http://www.smlib.com>
- [27] Cavalcanti, P. R., Carvalho, P. C. P., and Martha, L. F., 1997, "Non-manifold Modeling: An Approach Based on Spatial Subdivision," *Comput.-Aided Des.*, **29**, No. 3, pp. 209–220.
- [28] Marcheix, D., and Gueorguieva, S., 1997, "Topological Operators for Non-manifold Modeling," Proceedings of the 30th International Symposium on Automotive Technology and Automation, Mechatronics/Automotive Electronics, Florence, Italy, pp. 173–186.
- [29] Mantyla, M., 1988, "An Introduction to Solid Modeling," Computer Science Press.
- [30] Spatial Technology Inc., 1999, ACIS 3D Toolkit 5.0.
- [31] Wilson, P. R., 1988, "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.
- [32] Mantyla, M., and Sulonen, R., 1982, "GWB: A Solid Modeler with the Euler Operators," *IEEE Comput. Graphics Appl.*, **2**, No. 7, pp. 17–31.
- [33] Heisserman, J. A., 1991, "A Generalized Euler-Poincare Equation," Proceedings of the 1st ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, pp. 533.
- [34] Higashi, M., 1993, "High-quality Solid-modelling System with Free-form Surfaces," *IEEE Comput. Aided Des.*, **25**, No. 3, pp. 172–183.
- [35] Luo, Y., 1993, "Generalized Euler Operators for Non-Manifold Boundary Solid Modeling," *Geometric Modelling Studies 1990/3*, MTA SZTAKI, Hungary, pp. 19–34.
- [36] Luo, Y., and Gabor, L., 1991, "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, pp. 45–60.