Topological Operators and Boolean Operations for Complex-Based Non-Manifold Geometric Models

Hiroshi Masuda.

IBM Research, Tokyo Research Laboratory

Abstract

Non-manifold geometric modelling is used to represent and manipulate wireframe, surface, and solid models in a single architecture. It is suitable for improving geometric modelling environments for product design. In this paper, first, some fundamentals of non-manifold geometric modelling are presented. A mathematical framework is introduced and the topological structure and basic topological operations are discussed on the basis of the framework. Next, a new method, which can quickly and arbitrarily reshape geometric models defined by Boolean operations, is presented. This method is made possible by the capabilities of non-manifold geometric modelling, and can be used for design by trial and error and form-feature modelling.

keywords: non-manifold modelling, solid modelling, Euler operations, Boolean operations, form-feature modelling

Introduction

The concept of early manufacturing involvement is useful for realizing competitive products at low cost. In early manufacturing, it is very important to integrate design and process planning. By coupling design and process planning activities tightly, the feedback to the design phase from the process planning phase can be minimized. This concept is known as simultaneous engineering, or concurrent design. 3D geometric models, which represent the geometric shapes of products, play an important role in product design and manufacturing applications. They are also among the most important components of simultaneous engineering or concurrent design, because design activities are ultimately reducible to the geometric shapes of products. However, it is generally recognized that conventional 3D geometric models are not sufficient for realizing computerized systems that support complete design and manufacturing activities. This is mainly because they represent only the final shapes of products, and systems based on such models do not hold data on the meanings of the shapes, although this information would be useful in manufacturing applications. Therefore, when geometric models are used for various design and manufacturing applications, it is usually necessary to add some information that should have been given to the system when the model was created. In view of this, capturing and maintaining the meaning of a shape is very important.

Form-feature modelling [1, 2] is a promising concept for capturing the meaning of a shape. In this type of modelling, a product is described in vocabulary that is familiar to designers, and engineering meanings can be directly manipulated and maintained. In geometric modelling by means of form-features, the shape of a product is described as a set of subparts each of which has engineering meanings. Several modelling systems have been developed on the basis of the idea of the form-features. They have solved some problems that cannot be dealt with by conventional modelling systems, but fundamental problems are still left unsolved, because most existing form-feature modelling systems were developed by means of conventional solid modelling technologies. The main issues in conventional geometric modelling can be stated as follows:

- Form-features should be representable in any-dimensional form according to applications. In most commercial form-feature modelling systems, for example, a geometric model is constructed by using volumetric form-features, but multiple volumetric regions cannot be represented in a solid model, because of the limitation of the topological structure.

- Geometric shapes should be representable in wireframe, surface, and solid forms, because in the respective design phases, complete solid shapes are not always required. However, current geometric models cannot consistently represent multiple forms of geometric shapes in a single architecture.

- In current B-rep solid modelling, it is time-consuming to cancel or modify previous Boolean operations. It is of course easy to reshape models in CSG modelling, simply because intersections of primitive objects are not explicitly represented. However, CSG models are only useful for certain applications, and it is impossible to define shapes by using intersection lines and points [3]. Consequently, in current solid modelling techniques, it is difficult to create a 3D geometric model interactively by trial and error, especially when a 3D geometric model is defined by a large number of Boolean operations.

The idea of non-manifold geometric modelling [4, 5, 6] seems to be very promising for solving the above problems [7, 8], because

- Form features can be represented in wireframe, surface, and solid forms in a single architecture.

- Additional topological elements that do not appear in a resultant shape can be represented in a single geometric model.

- Boolean operations can be quickly reworked by using the method described in this paper.

We think non-manifold geometric modelling will make it possible to develop new geometric modelling systems that are more suitable for design.

To realize a non-manifold geometric modelling system, it is very important to investigate the characteristics of non-manifold geometric modelling minutely, because some conventional geometric modelling techniques are not adequate for non-manifold geometric modelling. It is essential to define *what is meant by a non-manifold geometric model*, because the term *non-manifold* is not a well-defined word, and it is possible to give various definitions of non-manifold geometric models [4, 7, 8, 9].

In this paper, first, we discuss a mathematical definition of non-manifold geometric models. On the basis of the definition, we then investigate the characteristics of non-manifold geometric modelling. We will discuss the topological structure and Euler operations for non-manifold geometric models. Next, we propose a method that can quickly and arbitrarily rework Boolean operations. This method makes it possible to create complicated solid models by trial and error. We also discuss the usefulness of this method for form-feature modelling.

Mathematical Definition

Non-manifold geometric models can be intuitively understood as combinations of wireframe, surface, and solid models. However, a mathematical definition is very important [10]. for defining the domain of non-manifold geometric models unambiguously. One suitable mathematical definition of non-manifold geometric models is *cell-complexes* that are subsets of 3D Euclidean space [7, 8]. Figure 1 shows some examples of cell-complexes. We call a geometric model defined as a cell-complex a *complex-based non-manifold geometric model*. The concept of cell-complexes is very suitable for representing geometric shapes that are meaningful in engineering, because it includes conventional wireframe, surface, and solid models, or combinations of them. In addition, since cell-complexes satisfy the Euler-Poincare formula, their Euler operations can be derived.

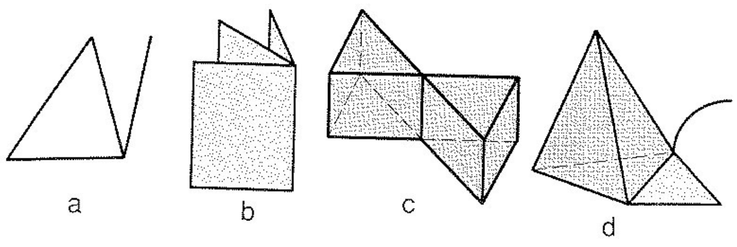

Figure 1: Examples of cell-complexes

Cell complexes are mathematically defined as sets of n-cells that satisfy conditions 1, 2, and 3. We define an n-cell as a bounded subset of 3D Euclidean space that is homeomorphic to an n-dimensional open sphere. A cell-complex and an n-cell are denoted by C and e_λ., respectively, and the dimension of e_λ and a closure of e_λ are denoted by $dim(e_\lambda)$ and $[e_\lambda]$., respectively.

1. $C = \bigcup_\lambda = \Lambda e_\lambda$
2. $[e_\lambda] - e_\lambda \subset \{e_\mu | dim(e_\mu) < dim(e_\lambda), \mu \in \Lambda\} \quad \lambda \in \Lambda$
3. $e_\lambda \cap e_\mu = \phi \quad \lambda \neq \mu, \lambda \in \Lambda, \mu \in \Lambda$

The first condition means that 3D cell-complexes can be represented by a collection of 0-cells, 1-cells, 2-cells, and 3-cells. In geometric modelling, 0-cells, 1-cells, 2-cells, and 3-cells can be called vertices, edges, faces, and volumes, respectively. The second condition means that the whole boundary of each element must consist of lower-dimensional elements. Because of this condition, a cell-complex is always closed. We do not think that this condition restricts the flexibility of geometric modelling, because unclosed, bounded edges or faces have no value in practical CAD applications. The third condition means that no topological element intersects another. This condition prohibits self-intersection. We believe that cell-complexes are suitable for geometric modelling in product design, and can cover geometric shapes that appear in CAD applications.

Data Structure

Topological Elements

In boundary representation solid modelling, an object is represented by a collection of faces, edges, and vertices, but in non-manifold geometric modelling, an object is represented by a collection of volumes, faces, edges, and vertices. We will refer to these as *topological elements.* A volume is defined as a bounded, open, continuous, solid space. The neighborhood of any point in a volume must be homeomorphic to a 3D open sphere. Figure 2 shows a solid cube and an empty cube. A solid cube is represented by defining a volume in a 3D space bounded by faces. When a volume is not defined, as shown in Figure 2b, the space inside the the box is regarded as a cavity.

When faces and volumes are defined as 2-cells and 3-cells, respectively, they cannot have cavities and through-holes, because of the definition of n-cells. However, cavities and holes frequently appear in mechanical products. Therefore, we allow faces with cavities and volumes with cavities and holes. Even if topological elements have cavities and through-holes, theorems on cell-complexes can be applied by dividing them into sets of n-cells. To represent topological elements with cavities, *shells* and *loops* are introduced, because a topological element with cavities has multiple disconnected boundaries. As shown in Figure 3, a loop consists of connected edges, or an

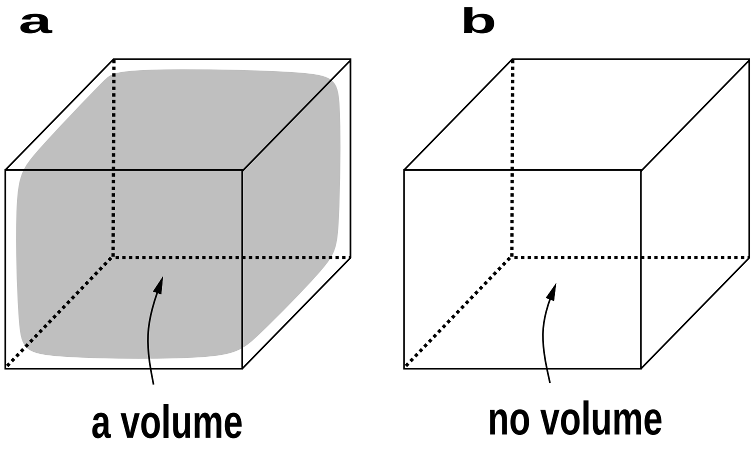

Figure 2: Representation of a solid cube and an empty cube

isolated vertex, and a shell consists of connected faces and wire-edges, or an isolated vertex.

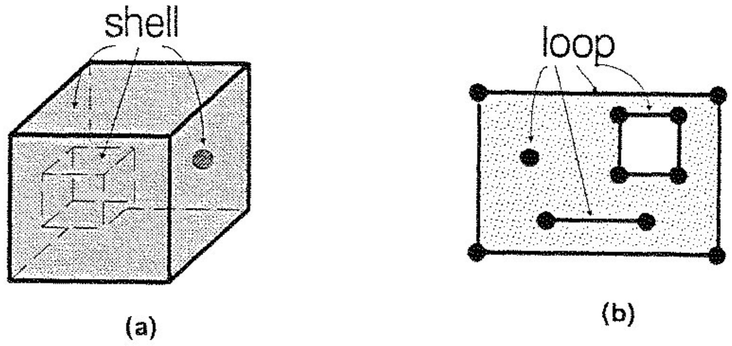

Figure 3: Representation of disconnected boundaries: (a) shells, (b) loops

Topological Structure

Topological elements are hierarchically interrelated. A lower-dimensional element is used as the boundary of each of several higher-dimensional ones. The hierarchical relationship of topological elements is shown in Figure 4. In this structure, a topological element can be linked with several types of higher-dimensional topological element. A face can be linked with a complex or volumes, depending on whether it is used as a laminar-face or as the boundary of one or two volumes. An edge can be linked with a complex, shells, or loops. If an edge is directly linked with a complex or a shell, it is used as a wire-edge in a 3D space or a volume. When an edge is linked with loops, it is used as the boundary of each of several faces. A vertex can be linked with edges, loops, a shell, or a complex. When a vertex is linked with loops, a shell, or a complex, it is used as an isolated vertex in a face, a volume, or a 3D space, respectively.

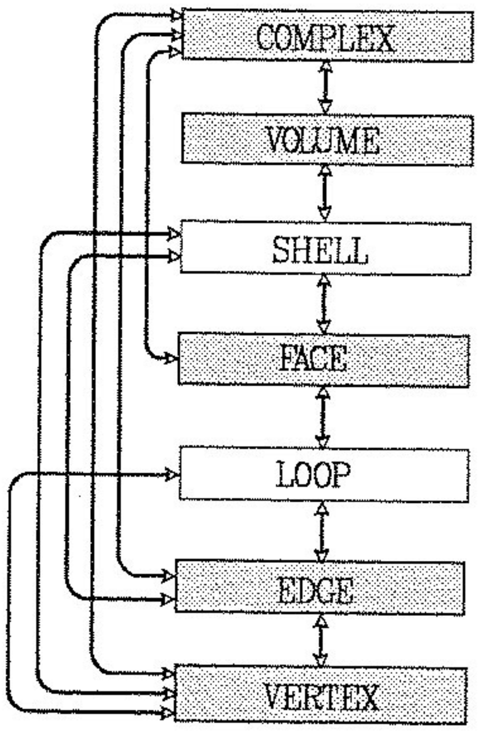

Figure 4: Hierarchical structure of topological elements

In order to manipulate topological elements efficiently, ordering information as well as adjacency information must be maintained. The order of edges around a face and of edges around a vertex are well-known examples of ordering information in solid modelling. They have been typically represented by the winged-edge structure or its variations. In non-manifold geometric modelling, however, it is also necessary to maintain the order of faces around an edge, which is called the radial-edge order [5]. Since radial-edge orders are very useful for detecting closed spaces bounded by faces, they should be explicitly maintained in order to manipulate geometric models with multiple volumes effectively.

Extended Euler Operators

Euler Operations

Topological operations are used to generate and modify the topological structure of geometric models. In solid modelling, the most popular topological operations are Euler operations [11, 12, 13]. Euler operators are known to have the following useful features:

- All high-level topological operators, such as generation of primitive objects or lift operations, can be defined by a sequence of Euler operations.
- Euler operators maintain the consistency of the topological structure.
- An Euler operator has an inverse operator. Therefore, if a high-level operator is defined by Euler operators, the inverse operator of the high-level operator can be easily obtained as a sequence of inverse Euler operators.

However, conventional Euler operators cannot manipulate non-manifold objects, because Euler operators are determined according to the following Euler-Poincare formula, but the formula is not satisfied by non-manifold objects:

$$v - e + f - r = 2(s - h) \tag{1}$$

where, v, e, and f are the numbers of vertices, edges, faces, respectively, r is the number of rings that are cavities in faces, s is the number of shells that are continuous surfaces, and h is the number of holes. However, this formula is based on the fact that a solid model has only one volume that is adjacent to all other topological elements. A counterexample of formula (1) is the object in Figure 1c. This object consists of 10 vertices, 17 edges, 10 faces, and one shell, and thus $v - e + f - r = 3$ and $2(s - h) = 2$.

Euler-Poincare Formula for Non-Manifold Geometric Models

It is known that cell-complexes satisfy the following formula, which is called the *Euler-Poincare formula for finite cell-complexes.* Here, α_i is the number of i-dimensional cells, and b_i is the i-dimensional Betti number of cell-complex objects.

$$\Sigma_{i=0}^{n}(-1)^i\alpha_i = \Sigma_{i=0}^{n}(-1)^i b_i \tag{2}$$

The formula for non-manifold geometric models can be obtained by dividing topological elements with cavities and holes into sets of n-cells and applying formula (2). The formula for non-manifold geometric models is as follows. This formula is satisfied by every non-manifold geometric model defined in this paper.

$$v - e + (f - r) - (V - V_h + V_c) = C - C_h + C_c \tag{3}$$

where v, e, f, V, and C are the numbers of vertices, edges, faces, and volumes, and complexes, respectively, r is the number of rings, Vh is the number of holes through volumes, Vc is the number of cavities in volumes, Ch is the number of holes through complexes, and Cc is the number of cavities in complexes. Since this formula is based on the Euler-Poincare formula for cell-complexes, we call formula (3) the *Euler-Poincare formula.*

We believe that this formula can be easily understood by application programmers, because it is described by means of topological elements, cavities, and through-holes, which are natural in engineering. Figure 5 shows applications of formula (3). In Figure 5a, the object has one hole topologically, because one closed path exists in the sequence of wire-edges. In Figure 5b, the volume has two cavities. One is an isolated vertex, and the other is a 3D empty space bounded by faces. In Figure 5c, the object has a non-manifold edge. In Figure 5d, the object consists of two volumes. One has a hole through the volume, and the other fits into the hole. Therefore, the space occupied by this object is a solid cube with no holes and no cavities.

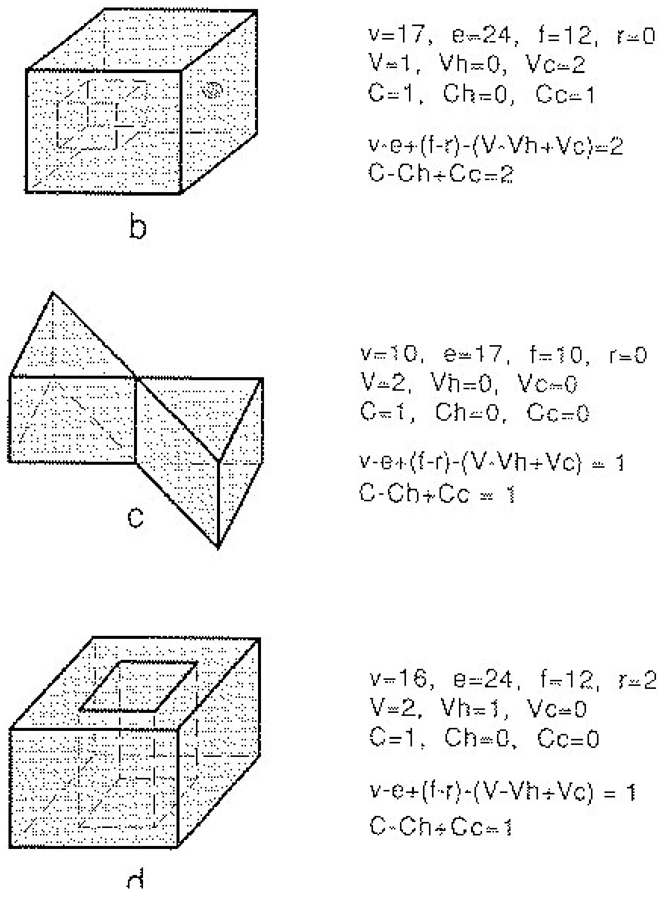

Figure 5: Examples of the application of the Euler-Poincare formula

Extended Euler Operators

Topological operators for non-manifold geometric modelling can be determined according to formula (3). We call these operators *extended Euler operators*. Theoretically, only nine independent Euler operators and their inverse operators are sufficient to define all complex-based non-manifold geometric models. In practice, however, more than nine Euler operators are needed, because it is not efficient to describe high-level operators with only nine Euler operators. The extended Euler operators used in our system are shown in Figure 6. In this figure, reverse operators (arrows to the left) are enclosed in brackets, and a cavity and a hole in a volume are denoted by Vcavity and Vhole, and a cavity and a hole in a complex are denoted by Ccavity and Chole.

High-level operators, such as local operations and Boolean operations, can be described by a sequence of Euler operators. The sequence of operations for a lift operation, for example, is shown in Figure 7. A lift operation modifies a shape by sweeping specified topological elements. In this example, a face and a wire-edge are swept, and a solid with a laminar-face is generated. In this operation, five kinds of Euler operator are used. In Figure 7b, wire-edges are generated from existing vertices, using *make_vertex_edge*, and in Figure 7c, wire-edges are generated between two vertices, using *make_edge_Chole*. In Figure 7d, five faces are generated by *make_face_kill_Chole*, and in Figure 7e, a face is generated, and a closed empty box is made by *make_face_Ccavity*. In Figure 7f, *make_volume_kill_Ccavity* defines a volume in a closed space, and a closed space becomes a solid.

Boolean Operations

Current Status in Reworking Boolean Operations

Boolean operations are among the most popular operations for making solid models of products. In modelling with Boolean operations, a new solid model is gradually determined by two intersecting solid models. Some initial solid models, such as blocks and cylinders, are provided by the system. They are called *primitive objects*. There are three types of Boolean operation, which are called union, difference, and intersection.

Boolean operations are very useful and are used in most solid modelling systems, but they have serious drawbacks: it is time-consuming to reshape a B-rep solid model defined by Boolean operations. In CSG modelling, of course, models are easily reshaped by modifying a CSG tree, but it is also time-consuming to obtain boundary data.

Figure 8 shows three types of process for reworking a B-rep model. In Figure 8a, the system maintains only the sequence of operations. In this case, even when only one previously executed Boolean operation is modified, all Boolean operations must be executed from the beginning. Therefore, it is difficult to create a complicated model interactively by trial and error.

In Figure 8b, Boolean operations are constructed by means of Euler operators. A Boolean operation and its reverse operator are referred to as Δ_n and Δ_n^{-1}. This type of modelling system can quickly undo Boolean operations by using their reverse operators [14]. However, undo-operations also have drawbacks. When a Boolean operation applied at an early stage must be canceled, it is time-consuming to reshape a model by using undo-operations, because the system must cancel almost all Boolean operations and reapply them. This is mainly because the data required for the reverse operators depend on the operation sequence.

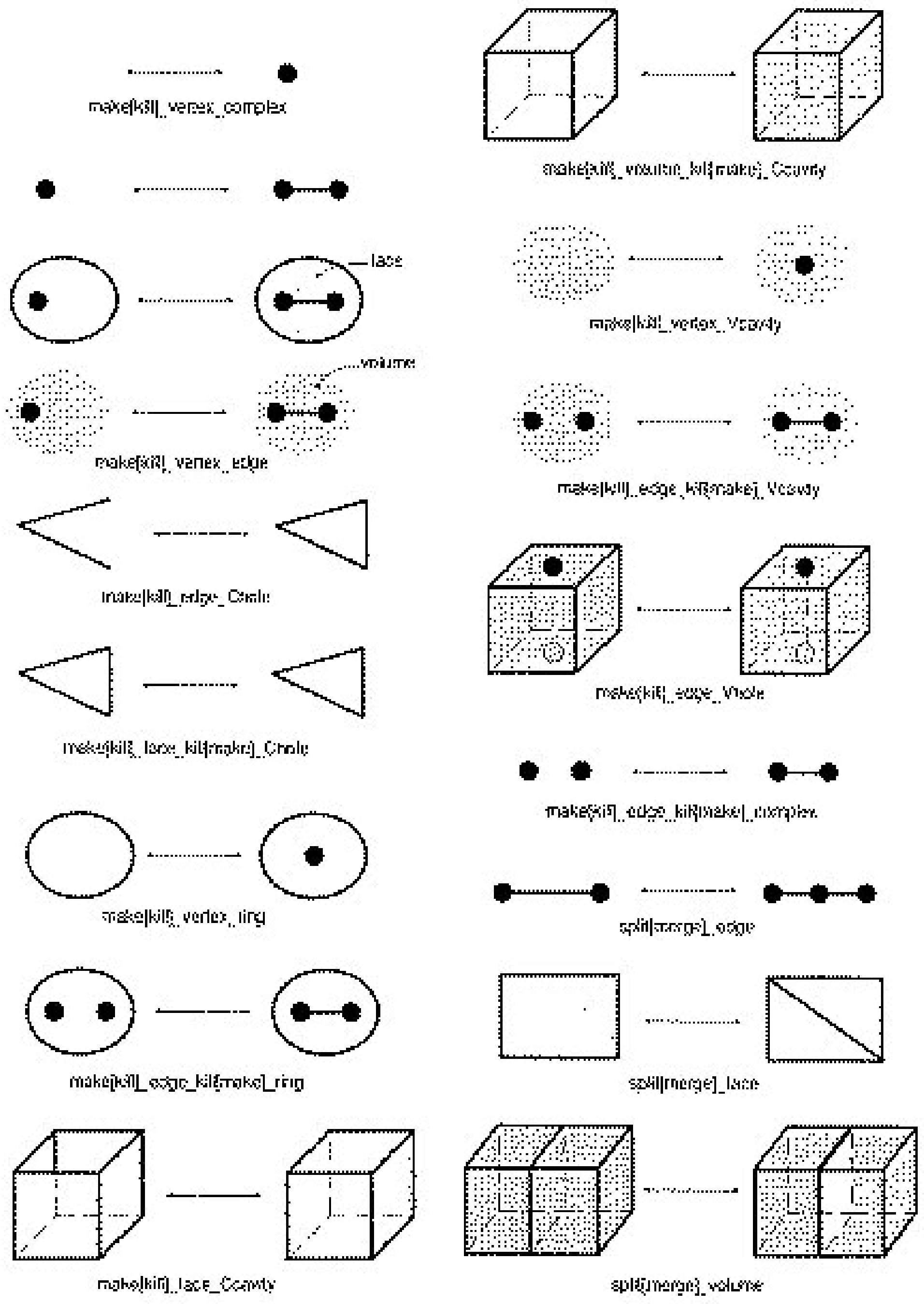
make[kill]_vertex_complex
face
volume
make[kill]_vertex_edge
make[kill]_edge_Chole
make[kill]_face_kill[make]_Chole
make[kill]_vertex_ring
make[kill]_edge_kill[make]_ring
make[kill]_face_Ccavity
make[kill]_volume_kill[make]_Ccavity
make[kill]_vertex_Vcavity
make[kill]_edge_kill[make]_Vcavity
make[kill]_edge_Vhole
make[kill]_edge_kill[make]_complex
split[merge]_edge
split[merge]_face
split[merge]_volume

Figure 6: Extended Euler operators

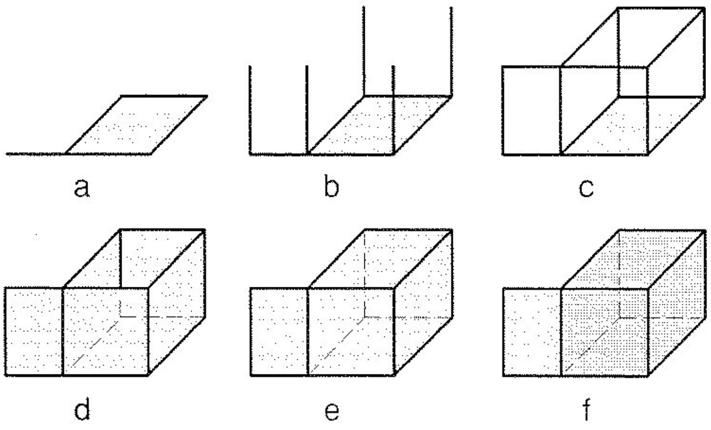

Figure 7: Lift operation: lifting a face and a wire-edge

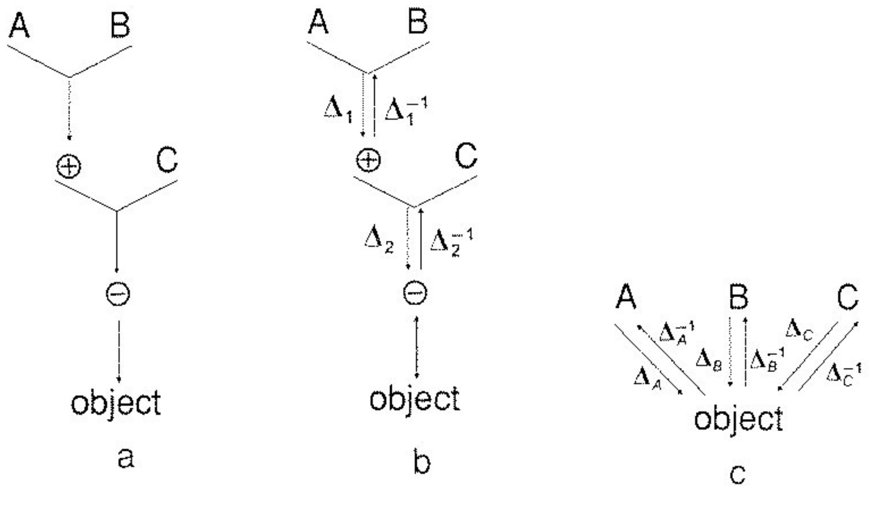

Figure 8: Three types of process for reworking a B-rep model: (a) reapplying all operations, (b) undo-operations, (c) cancel-operations

The most preferable reworking operation is a cancel-operation (Figure 8c). A cancel-operation reworks any Boolean operation at any time. Therefore, geometric models can be very quickly reshaped, even if they are very complicated objects. However, in current B-rep solid modelling, general cancel-operations are not realized. Therefore, when a solid model of a complicated object, such as an engine room, is modified, the model must be regenerated by a batch process. Cancel-operations are very important for interactive modelling of complicated objects.

Boolean Operations for Non-Manifold Objects

First, let us consider the definition of Boolean operations for non-manifold modelling. In complex-based non-manifold geometric modelling, the resultant shape of Boolean operations must satisfy the conditions of cell-complexes. However, the difference between two primitive objects may generate a shape without a boundary, as shown in Figure 9a. Therefore, we define a difference operation as the closure of the difference. Intuitively, the closure adds topological elements that make up for the lost boundary. The intersection, shown in Figure 9b, may generate wire-edges or lamina-faces that are degenerate forms of faces and volumes, respectively. This shape is a cell-complex, but wire-edges and lamina-faces are undesirable in most cases. On the other hand, however, lamina-faces and wire-edges might be useful in some applications, because they show where the two models touch. Therefore, intersection operations should have two types: one in which degenerate elements are retained (Figure 9b), and another in which they are eliminated (Figure 9c). Union operations, difference operations, and the two intersection operations are referred to as $\oplus$, $\ominus$, $\otimes^+$, and $\otimes^-$, respectively.

Boolean operations are basically dependent on the operation sequence. In general,

$$(A \oplus B) \ominus C \neq A \oplus (B \ominus C)$$

Obviously, Boolean operations do not satisfy the associative law or the commutative law. This is one of the main reasons that the realization of cancel-operations is difficult.

To solve this problem, we divided a Boolean operation into two operations: a *merging operation*, which is independent of the operation sequence, and an *extracting operation*, which is dependent on the operation sequence. Figure 10 shows the process of a Boolean operation. Here, we call the result of a merging operation a *merged object*, of which an example is shown in Figure 10b, and the result of a Boolean operation a *resultant object*, of which an example is shown in Figure 10c.

In merging operations, the system calculates the intersections of objects, and modifies the topological structure. Each topological element of a merged object has attributes that indicate the original primitive objects. These attributes are maintained every time the topological structure is modified. As shown in Figure 10b, a merged object maintains additional topological elements that do not appear in the resultant object. For this purpose, the capabilities of non-manifold geometric modelling are used.

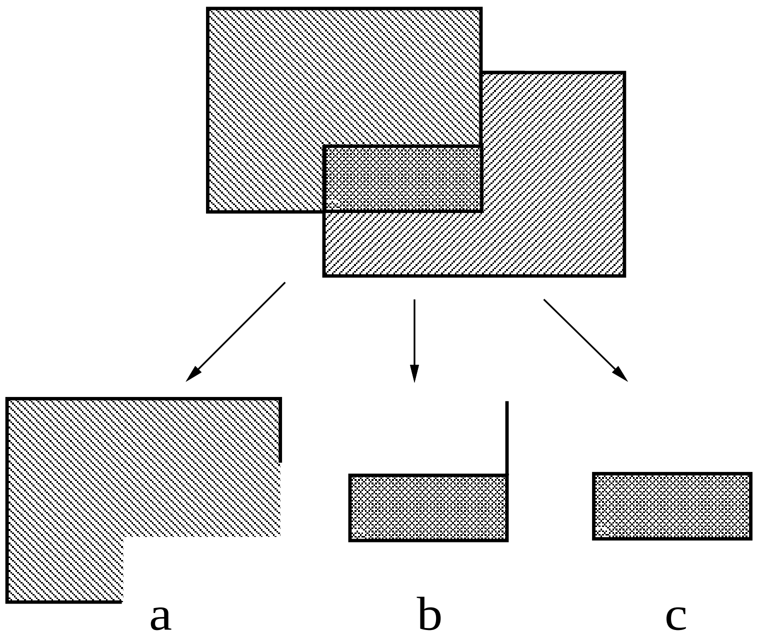

Figure 9: The difference and intersections between two primitive objects

The resultant object is obtained by an extracting operation. In extracting operations, the system extracts from the merged object topological elements that correspond to the resultant object, and adds marks to them. The execution time of this process is very short, because the system does not execute any geometric calculation or topological modification, but only traverses topological elements in a merged object. Extracted topological elements are, of course, dependent on the types and the sequence of operations. Figure 10c shows the three types of resultant object - union, difference, and intersection - that are extracted from the merged object.

Merging Operation

In conventional B-rep solid modelling, the original primitive objects are not left in the resultant shape. When two primitive objects are unified, for example, topological elements inside the united object are deleted. Therefore, the shapes of the original primitive objects are partly lost from the topological structure maintained by the system. When the system does not delete such topological elements, geometric models cannot be manipulated consistently in conventional solid modelling, because these geometric models have non-manifold conditions. The geometric model in Figure 10b, for example, has multiple volumes and non-manifold edges shared by four faces. Since these conditions can be manipulated by means of non-manifold geometric modelling techniques, it is possible to maintain the shapes of the original primitive objects in the topological structure.

Figure 11 shows an algorithm for a merging operation. In this example, A and B are merged. In Figure 11a, Fa and Fb are faces of primitive objects

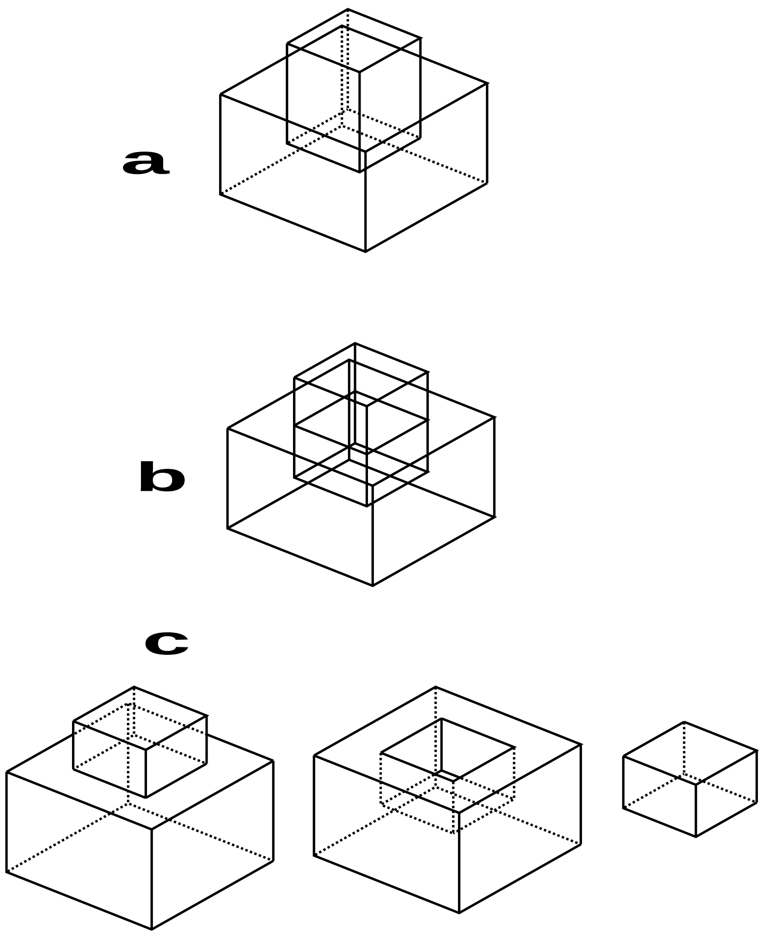

Figure 10: The difference and intersections between two primitive objects

A and B. All the topological elements are given attributes that indicate the original topological elements of primitive objects. In Figure 11b, the system generates intersecting edges and vertices on each object. If a topological element is split, its attributes are copied to a split topological element. In Figure 11c, the system merges the coincident topological elements of each object in the order of vertices, edges, and faces. The attributes of coincident topological elements are also merged. If a merged edge is shared by more than two faces, faces around the edge are sorted to maintain the radial-edge orders. When two models are merged, volumes may be split. The split volumes are reconstructed by detecting their boundary faces connected by the radial-edge pointers. Attributes of reconstructed volumes are also maintained.

The original primitive objects, which have been used to create the merged object, are maintained as sets of topological elements of the merged object. Figure 12 shows pointers between a merged object and primitive objects. Primitive objects have their original topological elements and a list of intersecting topological elements. In this figure, V_A and V_B are the original volumes of primitive objects A and B, and IE_{B1} and IE_{A1} are intersecting edges of A and B. V_1 , V_2 , V_3, and IE_1 are volumes and an edge in the merged object, respectively. When primitive objects are merged, the system maintains the relationships between the original topological elements and subdivided topological elements. When intersecting edges and vertices are generated, they are added to a list of intersecting topological elements. Since intersecting topological elements exist only when they are shared by two or more primitive objects, it is necessary to maintain the original topological elements and intersecting elements separately. Every topological element of the merged object must be related to at least one primitive object. In merging operations, the original topological elements are split or merged, but not deleted. Therefore, the relationship between a merged object and its original primitive objects can be easily maintained.

Extracting Operations

This section will describe the extraction of shapes defined by Boolean expressions. In merging operations, primitive objects are represented by sets of topological elements in the merged object. Therefore, we define Boolean operations among sets of topological elements. Suppose that the merged object is defined by two primitive objects, A and B, and that A and B are expressed by $\Lambda(A)$ and $\Lambda(B)$, which are sets of topological elements in the merged object. In $\Lambda(A)$, there are topological elements that are not used as the boundaries of other topological elements. Such topological elements are referred to as $\Lambda(A^{in})$; they include volumes, wire-edges, lamina-faces, and isolated vertices. $[\Lambda(A^{in})]$ expresses $\Lambda(A^{in})$ and their boundaries. Therefore, $[\Lambda(A^{in})]$ is equal to $\Lambda(A)$.

Topological elements of the resultant object are selected according to the type of Boolean operation. The resultant object is represented by the following topological elements. A union set, a difference set, and an intersection set are denoted by $+, -$, and $\cap$, respectively.

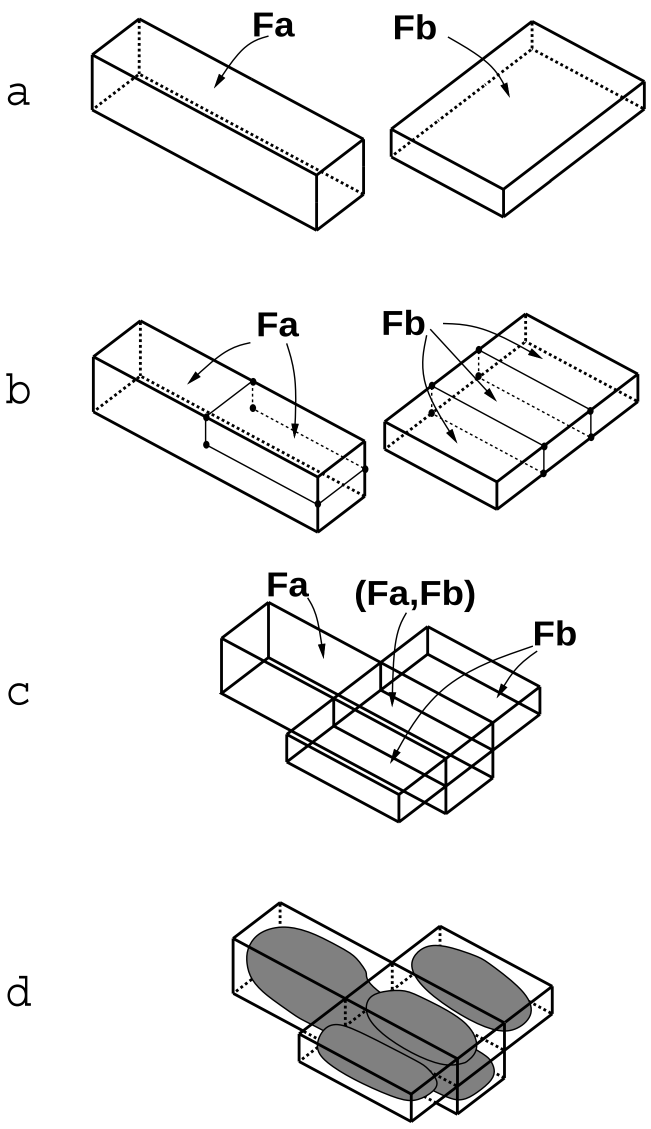

Figure 11: An algorithm for a merging operation: (a) two primitive objects, (b) generating intersecting edges and vertices, (c) merging coincident topological elements, (b) reconstructing volumes

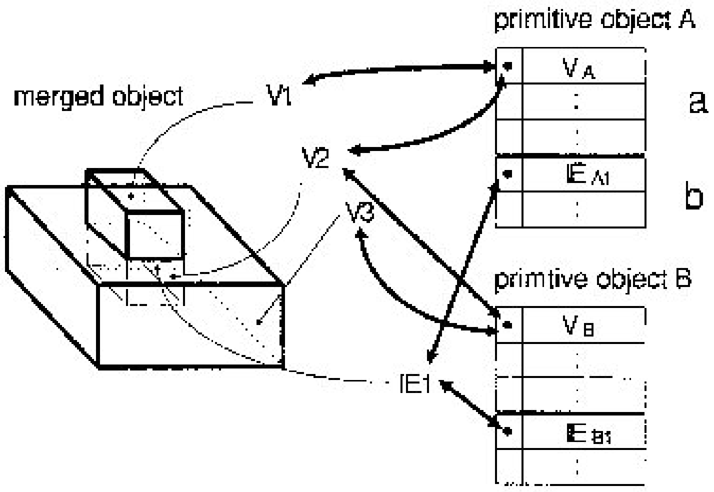

Figure 12: The relationships between the original primitive objects and the merged object: (a) a list of the original topological elements, (b) a list of topological elements generated by intersection

- $\Lambda(A \oplus B) = [\Lambda(A^{in}) + \Lambda(B^{in})]$
- $\Lambda(A \ominus B) = [\Lambda(A^{in}) - \Lambda(B^{in})]$
- $\Lambda(A \otimes^{-} B) = [\Lambda(A^{in}) \cap \Lambda(B^{in})]$
- $\Lambda(A \otimes^{+} B) = \Lambda(A) \cap \Lambda(B)$

Figure 13 shows an example for extraction. Model R in Figure 13b consists of the four cylindrical primitive objects shown in Figure 13a. It is obtained from

$$R = C \ominus D \oplus A \ominus B$$

Figure 13c shows a section of the merged object. V_i indicates the volumes in the merged object. The interior of each primitive object is
$\Lambda(A^{in}) = \{V_1, V_2, V_4, V_5, V_7, V_8\}$
$\Lambda(B^{in}) = \{V_2, V_5, V_8\}$
$\Lambda(C^{in}) = \{V_3, V_4, V_5, V_6, V_7, V_8\}$
$\Lambda(D^{in}) = \{V_3, V_4, V_5\}$
The resultant shape can be obtained from
$\Lambda(R^{in}) = \Lambda(C^{in}) - \Lambda(D^{in}) + \Lambda(A^{in}) - \Lambda(B^{in}) = \{V_1, V_4, V_6, V_7\}$
Shape R in Figure 13b is obtained by adding topological elements on the boundary of $\Lambda(R^{in})$.

Reworking Boolean Operations

It is very time-consuming to cancel or modify previous Boolean operations in solid modelling, because a Boolean operation is strongly dependent on the previous operations. If a Boolean operation can be canceled quickly

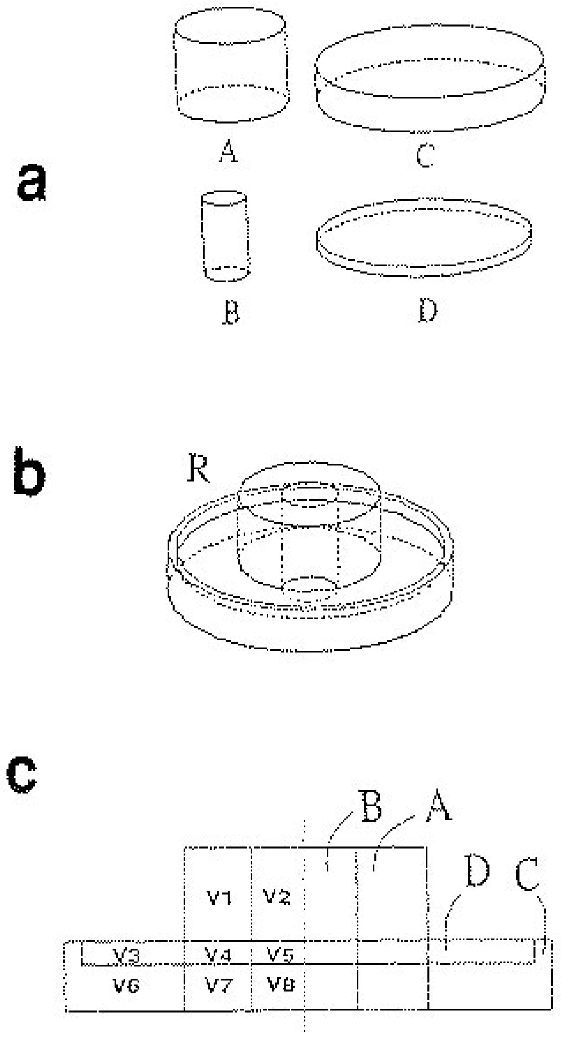

Figure 13: The resultant shape extracted from the merged object: (a) four primitive objects, (b) the resultant object, (c) volumes in the merged object

regardless of the operation sequence, a geometric model can be defined by trial and error.

When a Boolean operation is implemented by using a merging operation and an extracting operation, it is possible to remove or modify primitive objects very quickly. Since primitive objects are maintained as sets of topological elements in the merged object, the system knows which topological elements are to be deleted. To remove a primitive object from the merged object, the system reconstructs the topological structure simply by deleting the relevant topological elements pointed to by the primitive object. Deleted topological elements are either pointed to only by the removed primitive object, or generated by intersection between the removed primitive object and another one. This process is very quickly done by local modification. After the topological structure has been reconstructed, the system reevaluates the resultant shape.

Geometric models are often reshaped by modifying the shapes of original primitive objects. Even in this case, cancel operations are very effective. The reshaping process is as follows:

1. Primitive objects that are to be modified are removed from the merged objects by using cancel operations.

2. New primitive objects are merged into the merged object.

3. The new primitive objects are reintroduced at the same position in the Boolean expression, because Boolean operations depend on the order of operations.

4. The resultant shape is obtained by evaluating the new Boolean expression.

Figure 14a shows a merged object that consists of 150 primitive objects. Figure 14b shows the calculation time needed for the modification of primitive objects on an IBM RS6000/730. It took 50.36 seconds for 149 merging operations, and 0.16 seconds for an extracting operation. This reworking method is very effective, even if the models are very large.

Form-Feature Modelling

In the representation described in the previous section, any primitive object can be easily selected, deleted, or modified. These capabilities are useful in form-feature modelling, where an object is constructed from form-features, such as steps, chamfers, and grooves. When a form-feature is defined as a set of topological elements, it can be regarded as a primitive object. In this case, our modelling method can quickly display, delete, and modify any form-feature in the model.

We developed a prototype of a form-feature modelling system based on a non-manifold geometric modelling system. Our form-feature modelling system can solve the following problems:

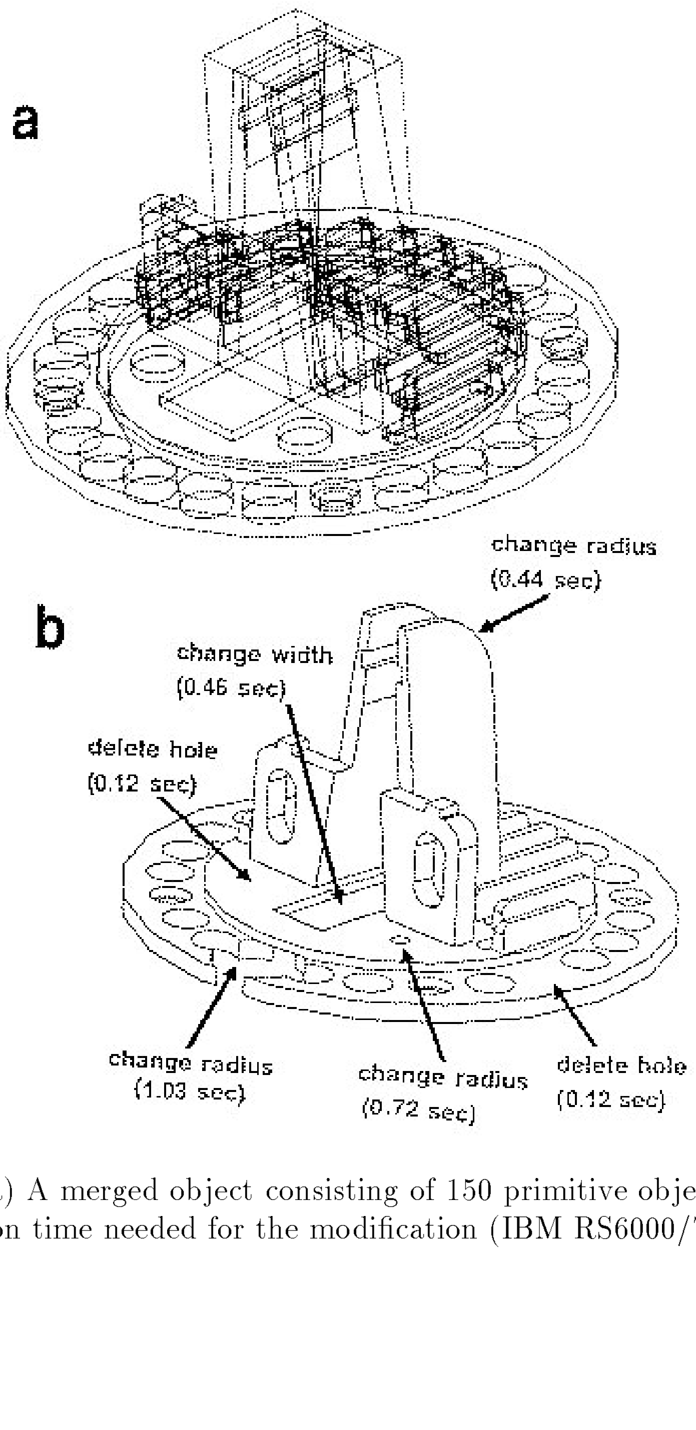

Figure 14: (a) A merged object consisting of 150 primitive objects, and (b) the calculation time needed for the modification (IBM RS6000/730)

- In conventional systems, it is time-consuming to obtain a resultant object by modifying existing form-features. In our system, form-features can be quickly modified.
- When form-features are maintained as sets of topological elements, it is difficult to maintain intersecting form-features, because the shapes of the form-features are partly lost, and their topologies are changed [15]. Our system can maintain intersecting form-features, because the lost parts of the form-features are maintained in the merged object.
- Many applications, such as constraint-based modelling, need to maintain the attributes given to topological elements throughout the whole design. Since the original topological elements of primitive objects are maintained by the system, their attributes are also maintained.

Conclusions

A non-manifold geometric modelling system is useful for improving 3D geometric modelling environments. Some important requirements for implementing such a system have been derived from the mathematical definition of non-manifold geometric models; it is adequate to define such models as cell-complexes. Extended Euler operators are useful topological operations for manipulating these topological elements consistently. One good application of non-manifold geometric modelling is in arbitrary reworking of Boolean operations. Our method can quickly reshape a B-rep solid model constructed by Boolean operations. It is suitable for design by trial and error, and is also useful in form-feature modelling.

Acknowledgement

We would like to express our gratitude to Dr. S.Kawabe for his useful advice, and to Messrs. K.Shimada, M.Numao, and S.Shimizu for their technical help with form-feature modelling. We would also like to thank Dr. M.Koda for his management support of this research project.

References

[1] P.R. Wilson and M.J. Pratt, 'A Taxonomy of Features for Solid Modeling' *Geometric Modeling for CAD Applications* North-Holland (1988) pp.125-136

[2] J.R. Dixon et al., 'Expert Systems for Mechanical Design: Examples of Symbolic Representations of Design Geometries' *Engineering with Computers* Springer-Verlag (1987) pp.1-10

[3] K. Shimada, M. Numao, H. Masuda, and S. Kawabe, 'Constraint-Based Object Description for Product Modeling' *IBM Research Report RT-0003* (May 1989), and *Computer Applications in Production and Engineering* North-Holland (Oct. 1989) pp.95-106

[4] K. Weiler, 'Topological Structures for Geometric Modeling' *PhD Thesis* Rensselaer Polytechnic Institute (Aug. 1986)

[5] K. Weiler, 'The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling' *Geometric Modeling for CAD Applications* North-Holland (May 1986) pp.3-36

[6] K. Weiler, 'Boundary Graph Operators for Non-Manifold Geometric Modeling Topology Representations' *Geometric Modeling for CAD Applications* North-Holland (May 1986) pp.37-66

[7] H. Masuda, K. Shimada, M. Numao, and S. Kawabe, 'A Mathematical Theory and Applications of Non-Manifold Geometric Modeling' *Advanced Geometric Modelling for Engineering Applications* North-Holland (Nov. 1989) pp.89-103

[8] S. Kawabe, K. Shimada, and H. Masuda, 'A Framework for 3D Modeling: Constraint-Based Description and Non-Manifold Geometric Modeling' *Organization of Engineering Knowledge for Product Modelling in Computer Integrated Manufacturing* Elsevier (Oct. 1988) pp.325-354, and *IBM Research Report TR87-1024* (Nov. 1988)

[9] J.R. Rossignac and M.A. O'Connor, 'SGC: A Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries,' *Geometric Modeling for Product Engineering* North-Holland (Sep. 1988) pp.145-180

[10] A.G. Requicha, 'Representations for Rigid Solids: Theory, Methods, and Systems' *ACM Computing Surveys* Vol. 12, No. 4 (Dec. 1980) pp.437-464

[11] B. Baumgart, 'A Polyhedron Representation for Computer Vision' *AFIP Conf. Proc.* Vol. 44 (1975) pp.589-596

[12] I.C. Braid, R.C. Hillyard, and I.A. Stroud, 'Stepwise Construction of Polyhedra in Geometric Modeling' *Mathematical Methods in Computer Graphics and Design* Academic Press (1980) pp.123-141

[13] M. Mantyla and R. Sulonen, 'GWB: A Solid Modeler with the Euler Operators' *IEEE Computer Graphics* Vol. 2 No. 7 (Sep. 1982) pp.17-31

[14] H. Chiyokura, *Solid Modeling with Design Base* Addison-Wesley (1988)

[15] M.J. Pratt, 'A Hybrid Feature-Based Modelling System' *Advanced Geometric Modeling for Engineering Applications* North-Holland (Nov. 1989) pp.189-202

[16] P.R. Wilson, 'Multiple Representations of Solid Models' *Geometric Modeling for CAD Applications* North-Holland (May 1986) pp.99-113

[17] M. Mantyla, *An Introduction to Solid Modeling* Computer Science Press (1988)

[18] C.M. Hoffmann, *Geometric and Solid Modeling* Morgan Kaufmann (1989)