The Principle of Duality in Data Structures and Euler Operators of Solid Modelers
(The Quarter-edge Data Structure)

Masatoshi NIIZEKI *1

*1 Osaka Electro-Communication University
Department of Electro-Mechanical Engineering
18-8 Hatsu-cho, Neyagawa-shi, Osaka 572-8530, JAPAN
niizeki@isc.osakac.ac.jp

Abstract
A data structure and a set of Euler operators for boundary representations of solid models in which the principle of duality is strictly achieved are presented. This new edge-based data structure is called the quarter-edge data structure. A data structure with complete symmetry between faces and vertices is derived from the quarter-edge. This data structure allows multiple loops of quarter-edges to belong to a single vertex. Euler operators based on the quarter-edge and this dual data structure make it possible to use the same code to implement two dual Euler operations. Duality in data structures and programs contribute to the robustness and efficiency in the implementation of solid modeling programs. These concepts can be extended easily to non-manifold solid models.
Keywords: geometric modeling , boundary representations, solid models

1 Introduction
There are many fields of study within geometric modeling where the principle of duality can be observed. The principle of duality between faces and vertices of the data structure of boundary represented solid models has been pointed out in many papers [1], [8], [6]. This duality in boundary represented solid modelers is also seen in Euler operators, which are the basic modification functions for boundary representation data structures.

Duality in geometric processing enables us to use a common structure for a pair of data elements and enables us to use common code for dual Euler operations. This duality makes programming efficient in size , execution speed and time required for development and maintenance. The theoretical duality of the data structure also guarantees the completeness and reliability of the program.

Although the principle of duality in the data structure and Euler operators of solid modelers has been noted in literature , not much research has been attempted to effectively exploit the underlying possibilities of using the principle of duality in solid modeling programs. For example, there seems to be very few solid modelers which use the same code for a pair of Euler operations. One reason for this is that duality in conventional data structures and Euler operators is incomplete in manifold solid models. Therefore, there has not been enough pursuit on the theoretical aspects of duality in data structures and Euler operations. The incompleteness of the duality in the data structure and Euler operations have been partially responsible for the inability to derive adequate data structures for non-manifold models. In order to fully incorporate the advantages of using the principle of duality in solid modelers, we must derive a more completely symmetric data structure.

This paper proposes a new edge-based data structure called the quarter-edge data structure which enables the programmer to have complete symmetry in Euler operators and some other basic solid modeling functions. A solid model representation based on this data structure is derived , and Euler operators are implemented for these solid models. This solid model fully utilizes the duality between faces and vertices in solid models. The topological data structure proposed in this paper can be implemented in a solid model combined with a geometric intersection computation and detection library based on the principle of duality between geometric entities, namely, points and planes.

2 Edge-based representation of solid models
2.1 Conventional edge-based data structures
(1) The winged-edge data structure
The winged-edge data structure (WE) is a commonly used data structure for representing solid models. The WE data structure is an edge-based data structure, which means that solid models are represented based on the connectivity of topological entities with respect to edges [2], [3]. A conceptual diagram of the WE data structure is shown in **Fig. 1**. The left part of **Fig. 1** illustrates the relative positions of the data elements, which are stored as pointers, in the solid model. Each WE stores pointers to the two faces and the two vertices adjacent to the edge, represented respectively as Face0, Face1, Vertex0, Vertex1 in the diagram. The other pointers are references to the WE which are adjacent to this WE. Wcw0 and Wccw0 are the Wes adjacent to this WE in a clockwise and counter-clockwise order along Face0. Similarly,

Wcw1 and Wccw1 are the Wes adjacent to this WE in a clockwise and counter-clockwise order along Face1. The right part of **Fig. 2** shows the pointers, which are contained in a single WE. The original WE data structure has problems representing solid models with curved surfaces because it is unable to distinguish between the different sides of the WE. A modified version of the WE data structure corrected that problem by adding information which distinguished which side of each WE is connected to this WE.

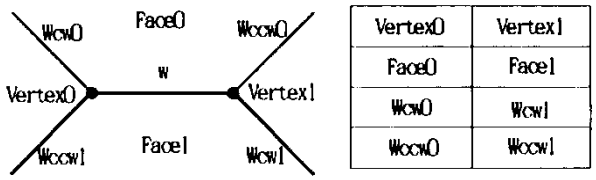

Fig. 1 Winged-edge data structure

It is easy to see that the WE data structure is symmetrical with respect to faces and vertices, that is, the structure retains information of faces and vertices in an exactly identical form. A single record of data which has this type of symmetry with respect to faces and vertices can be called a self-dual element. Although the WE is a self-dual data structure, the WE data structure was originally designed to be able to traverse edges along the boundary of a face. A WE represents one whole edge as a single unit of memory storage.

(2) The Half-edge data structure

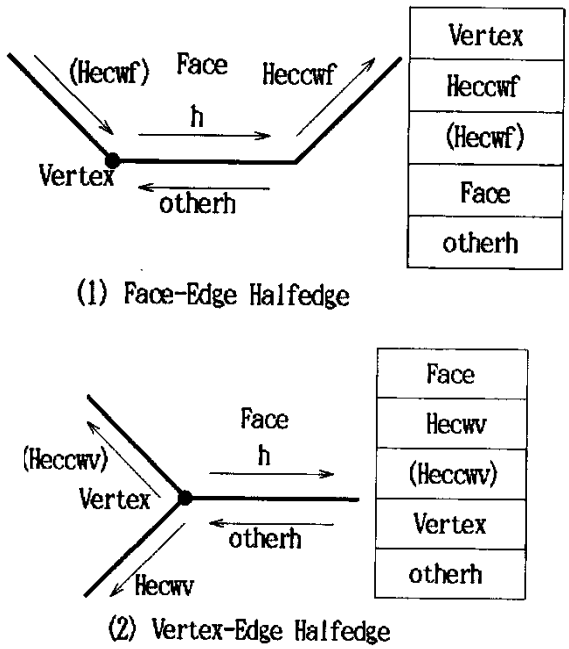

Fig. 3 Half-edge data structure

The half-edge data structure (HE) represents an edge in two separate parts [2], [3]. There are two distinct types of HE data structures. They are the face-edge (FE) type and the vertex-edge (VE) type. The FE type is mainly used for traversal of half-edges around faces, and the VE type is mainly used for traversal of half-edges around vertices. The field names are shown for both types of half-edges in **Figs. 2** and **3**, respectively. The fields shown in parentheses are not always necessary. These pointers are added for reasons of access efficiency.

These two types of HE data structures hold either face or vertex traversal information. The pointer otherh represents the HE which represents the pair half-edge belonging to the same edge to this HE. The principle of duality can be seen in the FE and VE half-edges. The FE structure and the VE structure are each other's dual entity. The main characteristic of both of these data structures is that it represents an edge in two separate parts.

Another type of HE data structure can be constructed by combining the FE and VE data structures. The combined half-edge data structure or FE-VE half-edge shown in **Fig. 4** simultaneously holds half-edge pointers for traversal around both faces and vertices. This data structure is very efficient from an access complexity point of view. It enables the efficient traversal of half-edges around faces and vertices. But its pointers are redundant and store more information than is absolutely necessary. Since this data structure stores face and vertex adjacency information in a symmetric fashion, it can also be considered a self-dual data structure. All of these half-edge types convey sufficient adjacency information for the representation of the complete topology of a solid model.

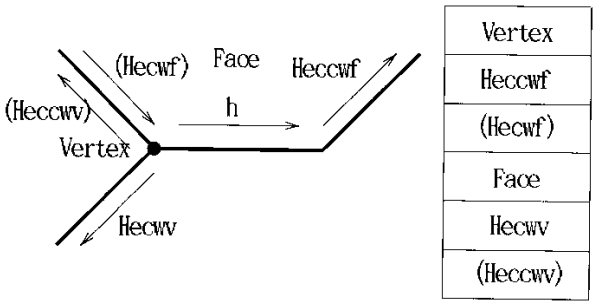

Fig. 4 Half-edge data structure (FE-VE combined type)

The only drawback of the half-edge data structure is the fact that the FE-VE half-edge is its own self-dual. In the case where face adjacency and vertex adjacency must be separated, the half-edge data structure cannot be used.

2.2 The quarter-edge data structure

There is an alternative way to hold separate FE type and VE type information in a solid model. If a half-edge is separated into FE and VE adjacency information, an edge is divided into four parts. This structure can be called the Quarter-edge data structure (QE), since edges are a combination of four QEs. The pointer usage of a QE is shown in **Fig. 5**. QEs are used as both FE type and VE type adjacency information. Although there are two separate uses of the QE, a single memory image can store the pointers for both uses. The principle of duality holds between the two uses of the QEs.

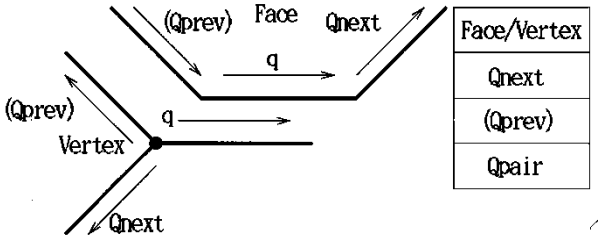

Fig. 5 Quarter-edge data structure

The field Qpair in **Fig. 5** store a pointer to a QE which is paired together with the original QE to act as on e FE-VE half-edge. The QE therefore has the same representation capability as the FE-VE half-edge.

One major advantage of the use of QEs to the other representations is that FE QEs and VE QEs can be specified as different entities. This is necessary if adjacency information must be specified as arguments to functions which can operate around both faces and vertices. For example, if an Euler operator is designed to take QEs as arguments, the Euler operator will have a different effect on a solid model depending on whether it takes a FE QE, or a VE QE. This functionality is necessary in order to program Euler operators for which the principle of duality holds. As will be seen in the following chapters, this cannot be done using self-dual edge based representations.

2.3 Comparison with other data structures

The QE data structure is designed in a symmetric fashion. The QE data structure is the only data structure that can be used to distinguish between face traversal and vertex traversal information. It uses slightly more memory than the FE-VE type half-edge. It also requires an extra step to switch between FE QEs and VE QEs.

3 The principle of duality in solid models

The principle of duality in data structures holds between faces and vertices. In order to create a solid modeling system which uses a single common structure for both faces and vertices, we will observe the basic operations for these elements.

As an example of a case where the principle of duality is applied so that a single source code can be used for two purposes, we examine the geometric computations and geometric intersection detections.

Duality can be observed in both the representation and the computations involving points and planes. A point in three-dimensional space is represented in homogeneous coordinates by four coordinates. A plane in three-dimensional space is represented in homogeneous coefficients with four components. This is the principle of duality in the representation of points and planes. Geometric intersection detection algorithms in three-dimensional space are combinations of sign detections of determinants. There are determinants of the homogeneous coordinates of points and determinants of homogeneous coefficients of planes. A procedure programed to compute the determinant of the homogeneous coordinates of points can also be used to compute the determinant of the homogeneous coefficients of planes. A procedure programmed to compute the determinant of the homogeneous coordinates of points can also be used to compute the determinant of the homogeneous coefficient of planes. Functions for points can be used as functions for planes without any modification. The difference in the resulting effect of the function is caused only by the difference in the interpretation of the input parameters. Functions of this kind can be used for two different purposes only when both the representation of the points and planes, and the operations on points and planes are identical within the program.

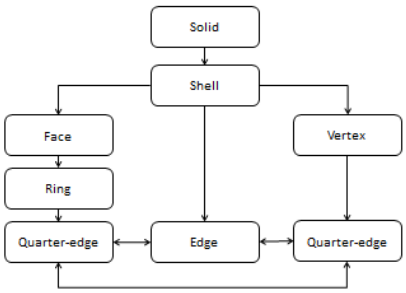

Fig. 6 Conventional solid data structure

Returning to data structures of solid models, the data structure in **Fig. 6** shows the dependence relationships of topological elements of a typical representation of non-manifold solid models. This representation in the figure is based on the quarter-edge introduced in previous sections, but representation based on Winged-edge or half-edge would be very similar in form. This figure illustrates how the symmetry in the data structure is incomplete because of a missing topological element on the right-hand side of the structure. There is no element on the right-hand side, corresponding to the rings on the left-hand side. The faces store pointers to rings while vertices store pointers to quarter-edges. The reference data in faces and vertices serve completely different purposes. Most procedures built for conventional data structures cannot be used for both faces and vertices without being modified because of this discrepancy.

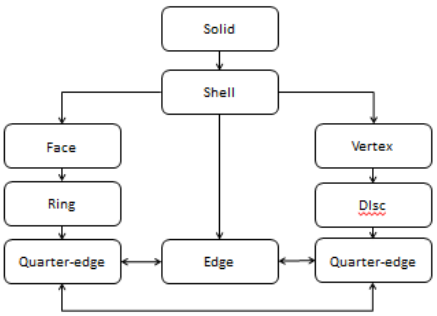

Fig. 7 Symmetrical solid data structure

A ring element in a solid model represents one closed loop of edges (or QEs) which form a part of the boundary of a face. Multiple rings are permitted to exist inside of a single face. In order to create a symmetric data structure, a new topological element must be added which serves as counterpart for the ring element. We call this element a disc. A disc element

represents a closed loop of edges (or QEs) which form a part of the boundary of a vertex. Multiple discs may exist inside of a vertex using this representation. Manifold solid modelers should only permit a solid to have one disc per vertex. An example of a non-manifold solid with multiple discs in a single vertex is shown in **Fig. 8**. There are three discs shown which belong to the central vertex.

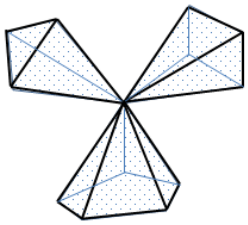

Fig. 8 Non-manifold solid represented by symmetrical solid data structure

The introduction of a disc element enables us to use faces and vertices in an equal way. A diagram of a solid model representation with the disc element is shown in **Fig. 7**. Observe the complete symmetry of the structure. The face and the vertex, the ring and the disc are just different uses of the same type of element. The reference information stored in each element is identical.

This representation is a superset of the conventional manifold solid model representation, so any manifold solid can be represented using this new symmetrical structure.

4 The principle of duality in Euler Operators

4.1 Dual Euler operators

The new data structure presented in previous sections has a new topological element. The modified Euler-Poincare equation for the solid models represented using this data is shown below.

$$(v-d)-e+(f-r)=2(s-h)$$

v: vertices d: discs
e: edge
f: faces r: rings
s: shells h: holes

This equation is a subset of the equation already established for non-manifold solid models in other papers. A set of Euler operators which operate according to this equation may as follows.

(1) mvfs (make vertex face shell), kvfs (kill vertex face shell)
(2) mef (make edge face), kef (kill edge face)
(3) mev (make edge vertex), kev (kill edge vertex)
(4) mekr (make edge kill ring), kemr (kill edge make ring)
(5) mekd (make edge kill disc), kemd (kill edge make disc)
(6) mfkrh (make face ill ring hole), kfmrh (kill face make ring hole)
(7) mvkdh (make vertex kill disc hole), kvmdh (kill vertex make disc hole)

The names of the Euler operators are shown along with their inverse operations. These fourteen operations are sufficient to create any solid model.

Conventional Euler operators for manifold solid models can be considered to be subset of the new Euler operators, so operations on solid models using conventional Euler operators can be implemented using the new Euler operators without any change.

The new Euler operators are built in pairs, according to the principle of duality. All of the Euler operators have corresponding operators obtained by interchanging the words “face” and “vertex”, “ring” and “disc.” The mvfs and kvfs operators are self-dual operators. The duality relationships between the Euler operators are shown below.

(1) mvfs, kvfs (self-duals)
(2) mef, kef < -- > mev, kev
(3) mekr, kemr < -- > mekd, kemd
(4) mfkrh, kfmrh < -- > mvkdh, kvmdh

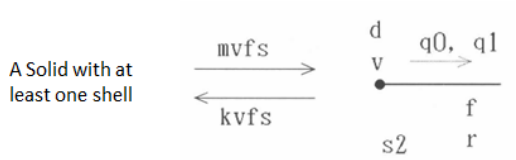

Fig. 9 mvfs, kvfs

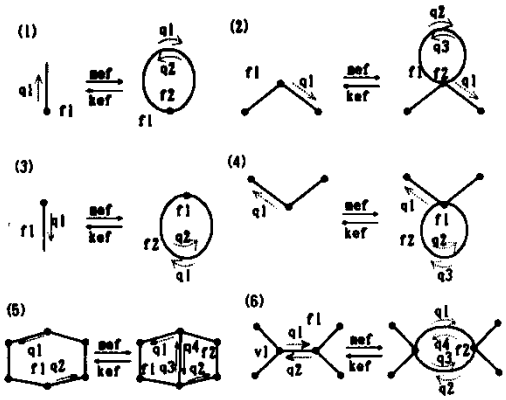

Fig. 10 mef, kef

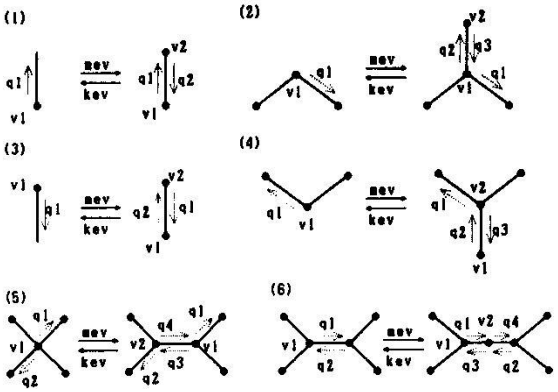

Fig. 11 mev, kev

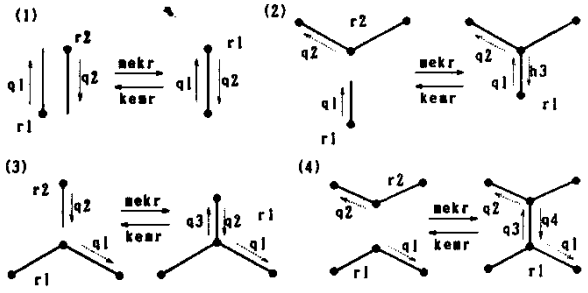

Fig. 12 mekr, kemr

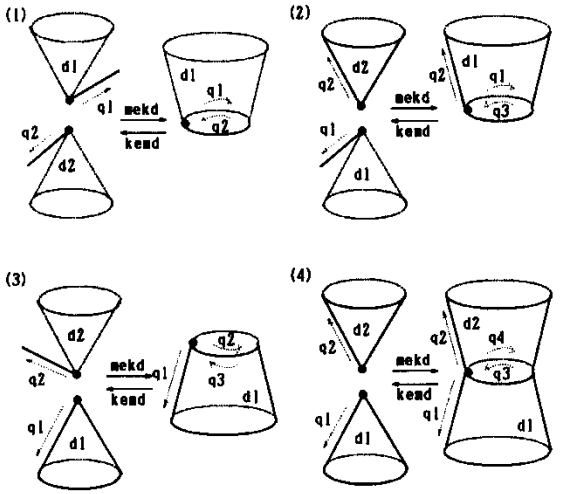

Fig. 13 mekd, kemd

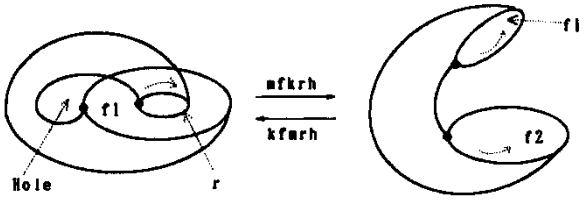

Fig. 14 mfkrh, kfmrh

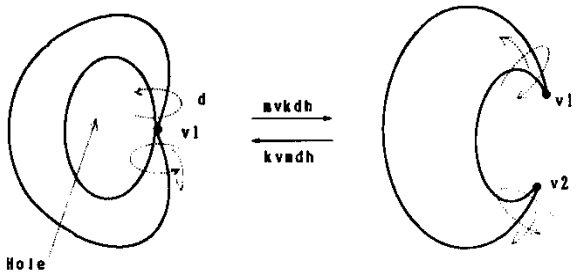

Fig. 15 mvkdh, kvmdh

Only one code for the functions for each of the pairs of Euler operators shown in the list is necessary. The action of each function changes according to the input arguments. For instance, one of the functions should act as either operator mef or mev depending on whether the inputs are face traversal or vertex traversal quarter-edges. WEs and HEs are insufficient for this purpose because they cannot distinguish whether they are to be used to specify face adjacency or vertex adjacency.

4.2 Basic data structure modifications by Euler operators

The principle of duality also holds inside of each of the new Euler operators implemented using quarter-edges and the symmetric data structure. Since a pair of Euler operators are implemented using a single function, each individual modification operation has two interpretations. This sections show the details of the actions of an implementation of each individual Euler operator. The Euler operators which are implemented using a single procedure are shown in pairs. The corresponding subprocedures in the pairs are given with the same number. Some of these procedures may seem unfamiliar. It is obvious from the principle of duality that every one of these operations is necessary to construct an arbitrary solid model. The duality of the Euler operators guarantees that there are no missing subprocedures. Complete Euler operators are obtained because of the principle of duality.

(1) mvfs (make vertex face shell), kvfs (kill vertex face shell)

(2) The Euler operator mvfs creates a new shell, face, vertex and a pair of quarter-edges. The operator kvfs is the inverse operation. **Figure 9** shows the result of the operation.

(3) mef (make edge face), kef (kill edge face) and mev (make edge vertex), kev (kill edge vertex)

(4) The operator mef creates a new edge and face inside of a face that already exists (**Fig. 10**). The operator kef is the inverse operation. The operator mev creates a new edge and vertex in an old vertex (**Fig. 11**) The operator kev is the inverse operation. Our implementation of mef and mev each provides six different ways to create new elements. The modifications in mef (kef) and mev (kev) are completely symmetric.

(5) mekr (make edge kill ring), kemr (kill edge make ring) and mekd (make edge kill disc), kemd (kill edge make disc)

(6) The Euler operator mekr deletes a ring by connecting two rings with an edge (**Fig. 12**). Two separate rings of quarter-edges are obtained from the operation. The operator kemr creates a new ring by removing an edge. On the other hand, the Euler operator mekd deletes a disc by connecting two discs with an edge (**Fig 13**). Two separate discs of quarter-edges are obtained from the operation. The operator creates a new disc by removing an edge.

(7) mfkrh (make face kill ring hole) , kfmrh (kill face make ring hole) and mvkdh (make vertex kill disc hole), kvmdh The Euler operator mfkrh removes a penetrating hole from a shell by creating a new fae from a ring (**Fig. 14**). The kfmrh operator is the inverse of mfkrh. The operator mvkdh removes a penetrating hole from a shell by creating a new vertex from a disc (**Fig. 15**). The kvmdh operator is the inverse operation.

5 Applications

5.1 Duality in primitive solid models

Most solid modelers provide some primitive solid models which can be created by specifying a small number of parameters. These primitive solids are modified and combined to obtain the intended model. Solid models created using the data structure and the Euler operators in the previous sections can be interpreted as two different solids. Therefore, each function for crating primitive solid models can be used for a pair of two different solids as can be seen in **Fig. 16**.

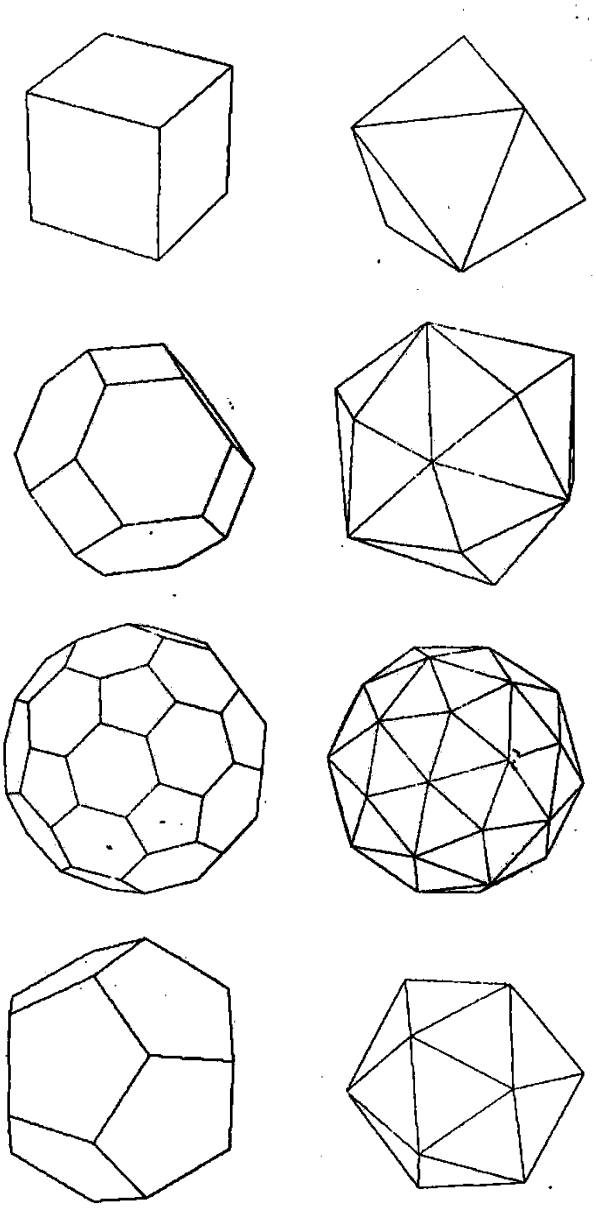

Fig. 16 Dual Shapes created by shared source code

5.2 Application to non-manifold solid models

Many methods have been proposed for the boundary representations of non-manifold solid models. The conventional manifold solid models do not necessarily require complete symmetry between faces and vertices. However, in a non-manifold modeler, the principle of duality can serve as a criteria for checking if the representation has sufficient capabilities. Research concerning non-manifold representations is often based on an extension of the Winged-edge or half-edge structure. The quarter-edge structure has the most powerful specification capabilities of any of the edge based data structures in that the quarter-edge can be used to distinguish between face adjacency and vertex adjacency.

6 Conclusions

The quarter-edge data structure has been proposed as a method of realizing the principle of duality in solid model representations. A symmetrical data structure combined with quarter-edge is necessary to implement Euler operators which can be used for dual operations. The quarter-edge representation is powerful in that face adjacency and vertex adjacency can be distinguished.

A simple solid modeling system with a set of dual Euler operators was implemented using this data structure. The data structure and Euler operators were used in much the same fashion as conventional ones. Although a quantitative comparison cannot be presented, the new representation and Euler operators are more theoretically sound and is more suitable for extension to non-manifold solid models.

References

[1] M. Mantyla, “A Note on the Modeling Space of Euler Operators,” Computer Vision, Graphics and Image Processing, 26, (1984) 45.

[2] F. Yamaguchi, M. Niizeki, ”A New Paradigm for Geometric Processing,” Computer Graphics Forum, Vol. 12, No. 3, Conference issue for EUROGRAPHICS ‘93, September 1993, pp. C-177 - C-188.

[3] K. Weiler, “Edge-Based Data Structures for Solid Modeling in Cured-Surface Environments,” IEEE CG&A, (1985), 21.

[4] K. Weiler, ”Boundary Graph Operators for non-Manifold Geometric Modeling Topology Representations,” Geometric Modeling for CAD Applications, M. J. Wozny, H. W. McLaughlin, J.L. Encarnacao eds., IFIP, (1988) 37.

[5] H. S. M. Coxeter, “Introduction to Geometry,” John Wiley and Sons, (1961).

[6] T. C. Woo, J. D. Wolter, ”A Constant Expected Time, Linear Storage Data Structure for Representing Three-Dimensional Objects, ”IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, No. 3, May/June 1984, pp. 510－515.

Received on December 30, 2013
Accepted on February 3, 2014