A Hierarchical Boundary Model for Solid Object Representation

LEILA DE FLORIANI and BIANCA FALCIDIENO
Istituto per la Matematica Applicata del Consiglio Nazionale delle Ricerche

A new hierarchical model for solid object representation is described. This model, called a *hierarchical face adjacency hypergraph* (HFAH), is based on a relational description of the object boundary, called a *face adjacency hypergraph* (FAH), which considers faces as the primary topological entities defining the object boundary. The HFAH consists of a hierarchy of FAHs describing the decomposition of the boundary of an object into form features. In this paper the HFAH is described together with its internal encoding structure. Two basic transformations, called *refinement* and *abstraction*, are defined on the hierarchical model; these allow effective and efficient modifications of the hierarchical boundary model.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representation*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*computer-aided design (CAD)*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Hierarchical data structures, solid modeling boundary representations, tree graphs

1. INTRODUCTION

Representation schemes for three-dimensional objects can be classified into two major categories: volumetric and boundary representations [13]. Volumetric representation techniques describe an object as a combination of primitive volumes. This class includes all the decomposition techniques related to the underlying space representation, such as occupancy arrays and octree encoding [12], and object-based techniques related to predefined primitive volume elements, which are not directly tied to the coordinate system, such as constructive solid geometry techniques [13]. Boundary (or surface-based) representation techniques describe solid volumes in terms of their enclosing surfaces. Such models can be called solid models when they completely describe the form and the extent of the individual oriented surfaces of the objects. They must also contain enough information to determine how the surfaces are joined together to form completely closed and connected volumes.

Authors' address: Istituto per la Matematica Applicata del Consiglio Nazionale delle Ricerche, Via L. B. Alberti, 4, 16132 Genova, Italy.

A boundary representation of an object is a geometric and topological description of its boundary, which is segmented into a finite number of bounded subsets, called *faces*. Each face is, in turn, represented by its bounding *edges* and *vertices*. Thus a boundary model is composed of three primitive elements: faces, edges, and vertices. Faces are contiguous surface areas of the volume enclosed by boundaries. Edges are the elements that, when joined together, form the closed face boundaries. Vertices are those points on the surface at which edges intersect. Two additional entities, the *shell* and the *loop*, can be defined as collections of primary entities. A shell in an object S is defined as a maximal connected set of faces of S. A loop on a face f of S is defined as a closed chain of edges bounding f.

Both geometric and topological information are associated with the individual topological elements. In a boundary model, however, a clear separation can be made between the topological and the geometric descriptions of an object owing to their different characteristics and uses. Topological information is concerned with the adjacency relations between pairs of individual entities. There are 25 element adjacency relationships, each one consisting of a unique ordered pair from the five topological entities [16]. Geometric data define the shape and the space location of each primitive entity. For example, faces might have a surface equation to describe their geometry, edges three-dimensional spline parameters, and vertices their Cartesian coordinates. A boundary model must contain complete geometric and topological object descriptions, which are not mutually independent, but equally important for its representation.

Boundary models are the most widely used representations of solid objects in many different application domains. They have some important properties that make their use within a geometric modeler very attractive. Boundary schemes have a wide applicability; they are complete and also unique, provided that a partition of the boundary into maximal faces is considered [14]. Furthermore, they are sensitive, in the sense that they are able to represent finer characteristics of an object. On the other hand, boundary representations are lacking in conciseness and efficacy because they describe all the features of an object at the same level.

To overcome these drawbacks, we propose a new surface-based model, which can be organized into a hierarchical form, called the hierarchical face adjacency hypergraph (HFAH). This structure is based on a face adjacency hypergraph (FAH) description of the object boundary [3] and consists of a hierarchy of relational models, in which the root gives a relational representation of the global shape of the object, while any other tree node provides a topological description of a form feature attached to its parent node.

This model highly improves the conciseness and the efficacy of classical boundary models, while maintaining properties such as wide applicability, sensitivity, and precision. The use of this model provides some important benefits. The model completely reflects the object design process, which is intrinsically hierarchical, since the designer defines an object representation either by top-down refinement of the global object shape or by bottom-up assembly of object features. Second, it can be successfully applied to represent an object with its form features in CAD/CAM systems [4]. The HFAH has no rigid hierarchical

organization. It can be reorganized by applying the abstraction and refinement transformations defined in this paper so as to adapt it to the design functional requirements.

The remainder of this paper is organized as follows. In Section 2 the definition of FAH is briefly reviewed. Section 3 is a description of the HFAH and a data structure encoding the boundary of an object in a hierarchical form. In Section 4 two basic transformations on the hierarchical structure, called refinement and abstraction, are presented; these allow effective and efficient local modifications of the hierarchical boundary model.

2. THE FACE ADJACENCY HYPERGRAPH

The solid objects we consider belong to the class of three-dimensional objects bounded by compact (closed) orientable two-manifold surfaces. A topological description of the boundary of manifold objects is a relational model, in which the topological entities defining its boundary are explicitly represented, together with their mutual adjacency relations. The most common form of relational boundary model is the so-called *boundary graph*, which is defined by the vertices and the edges of the object embedded in a surface and is usually represented as a planar projection in the form of a Schlegel diagram [5, 10].

By reflecting the intrinsic hierarchy among the geometric data, in which surfaces are considered as primary defining geometric entities, and curves and points as derived ones, Ansaldi et al. have defined a relational boundary model, called the *face adjacency hypergraph* (FAH), in which the nodes describe the object faces and the arcs and hyperarcs represent the relationships among the faces induced by the sets of the edges and vertices, respectively [2, 3].

In a boundary representation BS of an object S the sets ES and VS of its edges and vertices define two relations over the set FS of the faces of S. Two faces f_1 and f_2 are called *edge adjacent* if and only if there exists an edge in ES shared by f_1 and f_2. Also, two faces f_1 and f_2 are called *vertex adjacent* if and only if there exists a vertex in VS belonging to both f_1 and f_2. The edge adjacency relation and the partition of FS defined by the vertex adjacency relation can be represented as a hypergraph, that is, a graph in which the arcs may connect an arbitrary number of nodes [6].

The FAH of an object S is a labeled hypergraph defined as a triple $G = (N, A, H)$ such that:

(i) For every face f in FS there exists a unique node in N corresponding to f and labeled f.

(ii) For every edge e in ES shared by two faces f_1 and f_2, there exists a unique arc in A joining the two nodes f_1 and f_2 corresponding to faces f_1 and f_2, which is labeled e.

(iii) Let $F_v \subset FS$, the set of the faces of S incident to a vertex v of VS, and $N_v \subset N$, the subset of nodes of G corresponding to the faces of F_v. Then for every v in VS there exists a unique hyperarc in H, that connects the nodes of N_v and is labeled v.

(iv) The arcs in A incident to any node f of G are organized into an ordered sequence according to the order of the corresponding edges on the face of S represented by f (i.e., if e' and e'' denote two arcs of A incident to f, then e'

is an immediate predecessor of e'' if and only if edge e' is the immediate predecessor of edge e'' in the loop of edges of S bounding face f).

(v) The hyperarcs in H incident to any given node f of G are organized into an ordered sequence hf, which is the order of the corresponding vertices on the face of S represented by node f.

(vi) The extreme nodes of any given hyperarc v of G are organized into an ordered sequence nv, which is the order of the corresponding faces around the vertex of S represented by hyperarc v.

All of the three primitive topological entities describing the boundary of an object are represented by distinct elements of the FAH. According to the above definition, the FAH can correctly represent objects in the specified domain with the following restrictions:

(i) two faces may share at most one edge;
(ii) all the faces are simply connected.

Thus the FAH is a labeled hypergraph that may contain self-loops but no parallel arcs. Under these assumptions, if only objects homeomorphic to a sphere are considered, the FAH representation is dual with respect to the boundary graph. The FAH may be disconnected: Every shell of the object is represented by a distinct connected hypergraph component. Conversely, any of such components in the FAH describes a shell of the object. The FAH may correctly represent curved-faced objects as well, since self-loops are considered admissible.

In order to produce a relational boundary model of general objects bounded by compact, oriented, two-manifold surfaces, we extend the definition of the FAH as described below. Any arc of the FAH is labeled with the names of the two extreme vertices of the edge it represents. Hence, objects with faces sharing two or more edges can be described by the model, since the edge–vertex relation is explicitly encoded into the labels assigned to the arcs of G. Also, each arc is labeled with the names of the two loops to which the corresponding edge belongs, thus encoding the edge–loop relation. The arcs incident to any given node f of G are organized into different ordered sequences, each composed only of those arcs that have the same loop label and thus correspond to a distinct loop on face f. In this way the loop–edge relation is encoded in the model. In other words, all the edges bounding any given face f of S can be obtained from the FAH G of S by considering the arcs of G incident to f according to their loop labels and giving the ordered arc sequence within each set of such arcs having the same loop label. By attaching the vertex and loop labels to its arcs, the FAH becomes a labeled multihypergraph, which provides a complete topological description of any solid object bounded by a compact, oriented, two-manifold surface.

3. THE HIERARCHICAL BOUNDARY MODEL

Classical boundary representation schemes describe all the parts of the boundary of an object at one level of specification. In many applications, however, it is important to separate the representation of the object at the highest level of abstraction from its lower level details. Any solid object can be viewed as being composed of a main shape and a collection of *shape attributes* or *form features* [4, 9]. By form feature of an object S, we mean a region of interest on the surface

of S that affects the topology of the boundary model by modifying the number of its faces, edges, and vertices [17]. Examples of such features are protrusions, depressions, and through holes. Each feature can, in turn, be composed of a main structure plus lower level details, as occurs in the case of *compound* features. The recursive object decomposition into its main shape and a set of compound features defines a hierarchy in the object description. Using a boundary representation will produce a tree of boundary models.

Since any boundary scheme is based on a relational model connecting the three primitive boundary entities, a hierarchical boundary model is represented as a hierarchy of relational models, in which the root is the relational description of the main object shape and any other tree node provides a graph-based representation of a feature in its direct ancestor. In the following, we propose a hierarchical model based on the face adjacency hypergraph (FAH) and on the concept of hierarchical graph defined in [1], called the *hierarchical face adjacency hypergraph* (HFAH).

3.1 The Hierarchical Face Adjacency Hypergraph

A *HFAH* can be formally defined as a pair $g^* = (G, T)$, where T is the tree describing its hierarchical structure and G is a family of FAHs, each of which, called a *component* of g^*, is associated with a distinct node in T. The component G_0 corresponding to the root of T is called the *root component* of g^* and provides the FAH description of the main object shape. If G_i and G_j are any two components of g^* such that the node of T associated with G_i corresponds to the parent of the node associated with G_j, then G_i is called the *parent* of G_j and, conversely, G_j is called a *child* of G_i in g^*. Any nonroot component of g^* is the FAH representation of a feature in its parent graph.

The parent–child relation between any pair of components G_i and G_j of g^* is defined by a set of nodes that belong to both G_i and G_j. Any nonroot component G_j is connected to its parent G_i through a set of nodes C_j belonging to G_i, called *connection nodes* of G_j in G_i. G_i is thus called the *component of g^* defined by C_j*. The connection nodes of G_j in G_i correspond to those faces in the object to which the feature is attached. The nodes in the child graph G_j, which correspond to the nodes of G_i belonging to the set C_j, are called *dummy nodes*, since each dummy node in G_j describes a face added to the feature represented by G_j in order to form an admissible solid object. The set of the dummy nodes of G_j is denoted D_j. Hence, there exists a correspondence between the set D_j of the dummy nodes of G_j and the set C_j of the connection nodes of G_j in G_i expressed by a mapping q_j between C_j and D_j. The parent–child relation between G_i and G_j is thus defined by the triple (C_j, D_j, q_j). Figure 1b shows the hierarchical face adjacency hypergraph representation of the object in Figure 1a. The HFAH is composed of three face adjacency hypergraphs: G_0, which describes the main object shape, G_1, which represents the protrusion on face f_1, and G_2, which describes the cylindrical through hole attached to faces f_1 and f_2. In each diagram representing a component of the HFAH, dashed lines describe hyperarcs, and continuous ones denote arcs. The labels attached to the arcs are omitted for clarity. The relation between G_1 and G_0 is defined by the connection node f_1 in G_0, which is also a dummy node in G_1. The relation between G_2 and G_0 is defined by the pair $\{f_1, f_2\}$ of connection

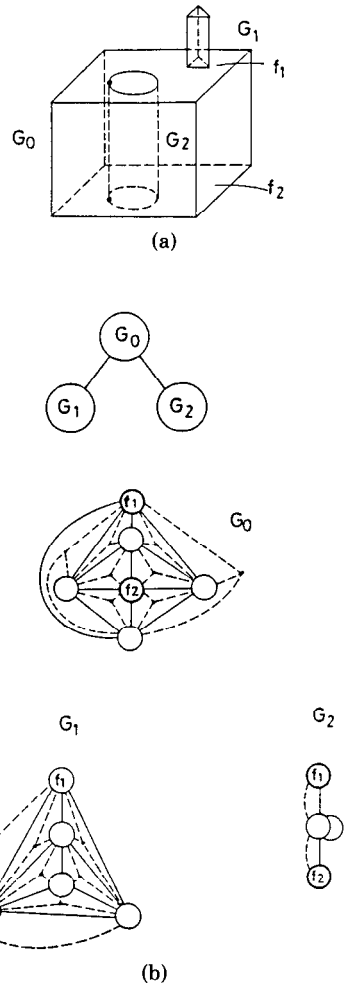

Fig. 1. Hierarchical face adjacency hypergraph representation of an object with two features.

nodes in G_0 that belong to G_2 as dummy nodes. Note that the arcs incident to f_1 and f_2 in G_0 correspond to the edges belonging to the external loop on the faces f_1 and f_2, respectively. The arcs of G_1 incident to the dummy node f_1 correspond to the edges of the interior loop defined on face f_1 of the object in Figure 1a by the protrusion. The same is true for the arcs incident to the two dummy nodes f_1 and f_2 of G_2.

Sometimes dummy edges and vertices must be added to the boundary of a feature in order to obtain a feasible object according to the domain specifications defined in Section 2. This is the case of the depression on an edge and of the bevel on an edge shown in the example of Figure 2. The depression on edge e must be completed by adding the two dummy faces, f_1 and f_2, and the dummy edge e' shared by the two faces. Component G_1 in the HFAH depicted in Figure 2b describes the resulting object, and G_0 represents the main object shape. Edge e' is described by the dummy arc e' in G_1 connecting the two dummy nodes

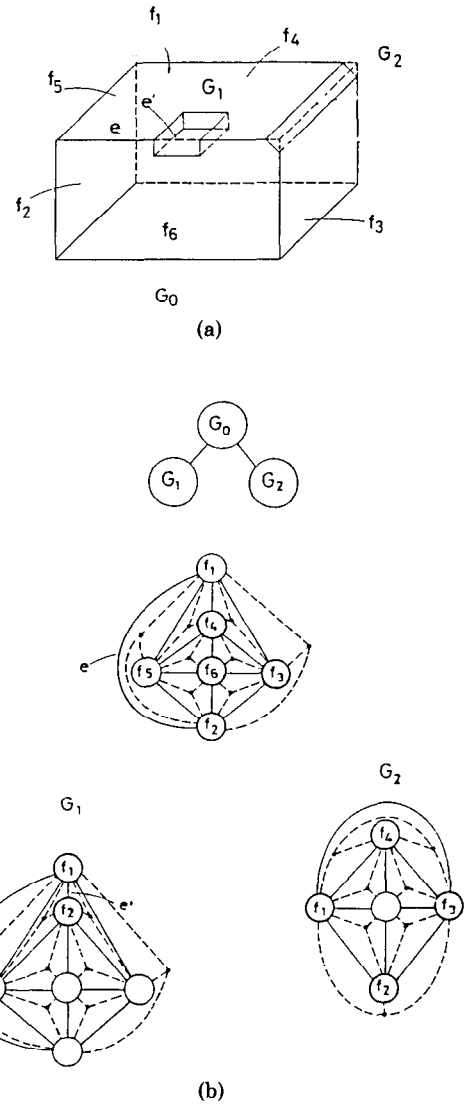

Fig. 2. Hierarchical face adjacency hypergraph representation of an object with a depression and a bevel.

f_1 and f_2. The face defining the bevel must be completed by adding four dummy faces, five dummy edges, and two dummy vertices, as shown by the dashed line segments in the drawing. The resulting object is described by component G_2 in the HFAH, which, henceforth, contains four dummy nodes, five dummy arcs, and two dummy hyperarcs. In general, any dummy edge or vertex added to a feature description is represented by a dummy arc or hyperarc in the corresponding component of the HFAH. Every dummy arc between two dummy nodes in a component G_j uniquely corresponds to an arc connecting the corresponding pair of connection nodes in the parent G_i of G_j. Similarly, every dummy hyperarc in a component G_j connects only dummy nodes and has a corresponding hyperarc in its parent G_i connecting the corresponding set of connection nodes.

As shown by the previous two examples, every dummy node in a component G_j corresponds to a connection node in its parent G_i. However, the loop of edges bounding any dummy face will be different from that bounding the corresponding connection face. The two loops may be completely disjoint, as in the case of the two features represented in Figure 1, or may have vertices or edges in common, as in the case of the depression and the bevel depicted in Figure 2.

As shown by the examples in Figures 1 and 2, a feature can originate inner loops belonging either to a single object face or to sets of adjacent faces. In the latter case we can group the set of connection nodes of a hypergraph G_i in the HFAH corresponding to an inner loop defined by the feature described by G_i into a single node that we call the *macroconnection node*. Similarly, the set of dummy nodes of G_i that corresponds to the set of the connection nodes forming a given macroconnection node will be grouped to form a *dummy macronode* f'. For simplicity, if f is a macroconnection node of G_j, we denote the corresponding dummy macronode of G_j by $q_j(f)$. The concept of macronode represents a way of structuring the sets of the connection and of the dummy nodes of a component of g^*.

3.2 A Hierarchical Boundary Data Structure

In this section we define a hierarchical data structure for encoding the boundary of a solid object, which is based on the HFAH. This hierarchical data structure is a combination of a tree and of the data structure, which encodes the face-based model. In this application we have adopted the symmetric structure [18]. This data structure explicitly encodes four relations, namely, the edge–face, the face–edge, the edge–vertex, and the vertex–edge relations. It is the most efficient in time and space complexity among the ones that explicitly encode the face–edge and edge–face relations, the basic relations for the FAH model. The resulting hierarchical boundary structure is described in Figure 3 in the form of a collection of Pascal record type declarations.

The data structure described in Figure 3 is capable of representing solid objects composed of a single shell. If an object contains hollow cavities, each shell will be represented by a different HFAH. The various HFAHs can be connected together into a linear list joining their roots or into a binary tree structure having nodes at their root components so as to maintain information on the containment relation among the object shells [15].

In the structure, any component G_i of the HFAH is described by a record of type *component*, which contains the pointer to the parent of component G_i (*parent*) and the link to the list of the faces of G_i (*facelist*). The list of the faces is organized as a singly linked list of records of type *face*. Each face record describing a face f contains the link to the next element in the list (*nextface*) and, if f is a dummy face, the pointers to the corresponding connection face in its parent component (*connectionface*) and to the first element of the sublist of the edges bounding f (*bedgelist*). If f is not a dummy, such a record contains the pointer to the list of the loops bounding f (*looplist*) and an additional pointer to the list of the child components having f as a connection face (*childlist*). This field will contain a nonnull value only if f is a connection face. The children of a

```
type        coptr       = ^component;
            fptr        = ^face;
            cptr        = ^child;
            lptr        = ^loop;
            beptr       = ^boundingedge;
            eptr        = ^edge;
            vptr        = ^vertex;
            ieptr       = ^incidentedge;

          component     = record
                              parent: coptr;
                              facelist: fptr
                          end;
          face          = record
                              nextface: fptr;
                              case dummy: boolean of
                              True: (connectionface:fptr;
                                     bedgelist: beptr);
                              False: (looplist: lptr;
                                     childlist: cptr)
                          end;
          child         = record
                              nextchild: cptr;
                              childcomponent: coptr;
                              dummyface: fptr
                          end;
          loop          = record
                              nextloop: lptr;
                              edgelist: beptr
                              face: fptr;
                          end;
          boundingedge  = record
                              nextbedge: beptr;
                              edge: eptr
                          end;
          edge          = record
                              loop: array [0..1] of lptr;
                              vert: array [0..1] of vptr;
                              dummy: boolean
                          end;
          vertex        = record
                              incidentedgelist: ieptr;
                              case dummy: boolean of
                              True: (parentvertex: vptr);
                              False: (x,y,z: real)
                          end;
          incidentedge  = record
                              nextedge: ieptr;
                              edge: eptr
                          end;
```

Fig. 3. Pascal record type declarations describing the hierarchical boundary data structure.

component attached to a given connection face f are organized into a linear list linked to the record describing f. Each element of this list (of type *child*) contains the link to the next element (*nextchild*) and the pointers to a child component (*childcomponent*) and to the dummy face corresponding to f in this child

component (*dummyface*). Note that each child of a component G_i appears in the child list of each of its connection faces in G_i.

The list of the loops bounding each face is a singly linked list, in which each record contains the pointer to the next element (*nextloop*), the link to the sublist of the edges forming the loop (*edgelist*), and the pointer to the record describing the face to which the loop belongs (*face*). The list of the bounding edges is a linear list, in which each record (of type *boundingedge*) contains the link to the next element in the sublist (*nextbedge*) and the pointer to the record describing the edge (*edge*). This substructure encodes the face–edge relation as a combination of the face–loop and loop–edge relations. Each edge of the object is encoded in a record of type *edge*, which contains the pointers to the loops sharing it (*loop*) and to the records describing its extreme vertices (*vert*), and a Boolean field (*dummy*), which specifies whether the edge is a dummy or not. Thus a record of type *edge* encodes both the edge–loop and the edge–vertex relations. The edge–face relation is encoded as a combination of the edge–loop and loop–face relations. Each vertex v is encoded in a record of type *vertex*, which contains the link to the sublist of the edges concurrent to it (*incidentedgelist*). Further, if v is a dummy vertex, the vertex record will contain a pointer to the record describing the corresponding vertex in the parent component (*parentvertex*); otherwise it will contain the Cartesian coordinates of v (x, y, z). Each sublist of incident edges is a singly linked list, in which each record (of type *incidentedge*) contains the pointer to the next element in the list (*nextedge*) and the pointer to the record describing the corresponding edge (*edge*). These sublists, together with the field *incidentedgelist* in each record of type vertex, encode the vertex–edge relation.

Since we are primarily concerned with the topological description of the object boundary, the three-dimensional coordinate values are the only geometric information that we have included in the structure. The three fields in each vertex record containing such information will therefore not be counted in the computation of the storage cost. The structure, however, could be extended to represent curved-faced objects as well, by associating the surface equation or patch information with every face record, and spline or other curve information with every record describing an edge [7, 15].

The storage complexity of the hierarchical boundary structure of Figure 3 can be evaluated in terms of the number of object faces (F), loops (L), edges (E), and vertices (V), of the number of structure components (N) and of the total number of dummy faces (F_d), dummy edges (E_d), and dummy vertices (V_d) in the whole structure. Hence, the number of records of type *component* is N, and we have exactly $F + F_d$, $E + E_d$, and $V + V_d$ records of type *face*, *edge*, and *vertex*, respectively. Since each record in the list of the child components corresponds to a different dummy face, the total number of such records is equal to F_d. The storage cost of the list of the records describing the object edges (records of type *edge*) is equal to $4(E + E_d)$, whereas encoding the loop–edge (records of type *boundingedge*) and the vertex–edge (records of type *incidentedge*) relations requires a number of fields equal to $8(E + E_d)$. Thus the total storage cost of the resulting structure is equal to

$$2N + 3F + 6F_d + 3L + 12E + 12E_d + V + 2V_d. \tag{3.1}$$

The storage cost deriving from encoding the object boundary as a "one-level" symmetric structure is equal to

$$3F + 3L + 12E + V. \tag{3.2}$$

If we compare the previous two expressions, we find that the overhead due to the hierarchical organization, and thus to the decomposition of the object into its features, is equal to

$$2N + 6F_d + 12E_d + 2V_d, \tag{3.3}$$

which depends only on the number of structure components and on the connections between any pair of them defined by the dummy elements, and thus on the complexity of the object decomposition and its features.

The time complexity of the hierarchical boundary structure is evaluated in terms of the structure accessing algorithms that retrieve information about the mutual adjacency relations between pairs of primitive entities. Queries on those adjacency relations involving entities that belong to the same component, and neither are dummy elements nor belong to a connection or a dummy face, can be answered by simply examining the symmetric structure describing that component. Hence, the same accessing algorithms as those defined in [18] for the symmetric structure can be applied. The time complexity will be the same as that of the symmetric structure; that is, $5EV_i + 4K$. If the hierarchical boundary model is composed of components that contain only dummy faces, then all the children of a component must be accessed in the worst case in order to answer any adjacency-finding queries. This affects the time complexity of the resulting structure by only a constant factor. In the general case, however, retrieving all of the nine adjacency relations may require the complete traversal of the hierarchical structure and, hence, is equivalent to the process of reconstructing the one-level expanded object representation from its hierarchical model. This process is based on a recursive application of a refinement transformation to each pair of parent–child components, as described in Section 4. Thus the worst-case time complexity of the hierarchical boundary model in a complete general case is equal to $F_d(5EV_i + 4K)$.

4. BASIC TRANSFORMATIONS ON THE HFAH

A hierarchical description of the boundary of an object can be built in different ways and can be used for different purposes. For example, a designer may define the HFAH of an object by specifying the main shape of the object at the highest level and inserting its lower details at different levels. In another application, an automatic procedure for feature recognition and extraction can build the HFAH according to the hierarchy of manufacturing features [9]. It could be necessary to convert the hierarchical description of an object defined in terms of design features into a description in terms of features of different functionality. For this reason two basic transformations, *refinement* and *abstraction*, can be defined on the HFAH representation of a solid object. These transformations, defined in [1] for general hierarchical hypergraphs, allow efficient local modifications of the hierarchical structure, since they operate on single pairs of components at the same time.

4.1 Refinement

A refinement transformation applied to an object S represented by a HFAH g^* consists of inserting the boundary description of a feature described by a component G_j of g^* as a part of the boundary representation of the feature described by its parent G_i in g^*.

In terms of the HFAH description, a refinement transformation applied to a component G_i and to one of its child components G_j consists of merging G_j into G_i and deleting G_j as a component of g^*. The merging operation is performed by "gluing" each connection node of G_j in G_i and the corresponding dummy node of G_j. This step is different, depending on the feature represented by G_j, since all possible dummy arcs and hyperarcs of G_i must be conveniently eliminated as a consequence. The deletion of G_j from g^* modifies the hierarchical structure of g^*, since all direct descendants of G_j in g^* are transformed into children of G_i.

An example of application of a refinement transformation is shown in Figure 4. Figure 4b depicts the HFAH representation of the object S in Figure 4c: Component G_0 describes the main object shape, G_1 the through hole, G_2 the depression on face f_1, and G_3 the depression on face f_3. By applying a refinement transformation to the two components G_0 and G_2, we obtain the HFAH representation of S depicted in Figure 4a: G_2 has been merged into G_0, and G_3 becomes a child of G_0.

Algorithm REFINE given below is a description of the effect of a refinement transformation applied to two components G_i and G_j of a HFAH g^* such that G_j is a direct descendant of G_i. The following functions and procedures, which operate on the tree describing the HFAH, are used as primitives in the algorithm description:

—CHILDREN (g^*, G_j): returns the collection of the direct descendants of G_j in g^*.

—ADD_ARC_HFAH (g^*, (G_i, G_j)): adds a new arc (G_i, G_j) to the tree describing g^*.

—DELETE_COMPONENT_HFAH (g^*, G_j): deletes a component G_j from g^*.

Algorithm REFINE (g^*, G_i, G_j, C_j, D_j, q_j);
// g^* is the HFAH description of the object;
G_i and G_j are the two components of g^* to which the refinement transformation is applied;
C_j is the set of the connection nodes of G_j in G_i;
D_j is the set of the dummy nodes of G_j;
q_j is the mapping from C_j to D_j which defines the relation between G_i and G_j //

1: GLUE (G_i, G_j, C_j, D_j, q_j);

2: **for every** component G_k **in** CHILDREN (g^*, G_j) **do**
ADD_ARC_HFAH (g^*, (G_i, G_k))
end for;
DELETE_COMPONENT_HFAH (g^*, G_j)
end REFINE.

Algorithm GLUE described below merges hypergraph G_j into G_i by identifying each node f in G_i belonging to C_j with node $q_j(f)$ in G_i. The dummy arcs and hyperarcs of G_j are deleted, new arcs are added to G_i corresponding to the edges split by the feature represented by G_j, and the vertex and loop labels attached to

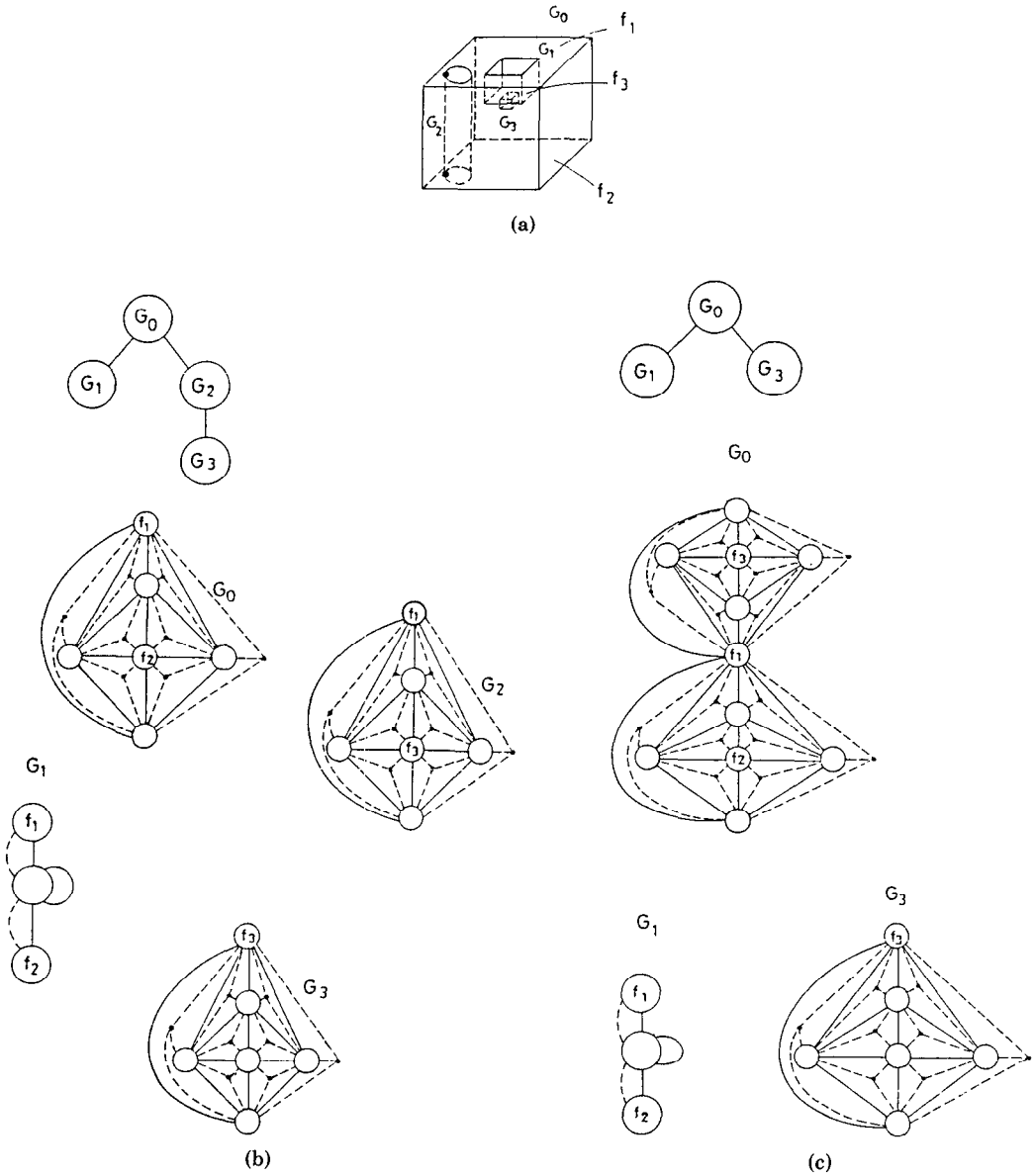

Fig. 4. (a) An object S with a through hole and two "nested" depressions. (b) HFAH representation of S in which each feature is described by a separate component. (c) HFAH representation of S obtained by applying a refinement to components G_0 and G_2 of the HFAH in (b).

arcs are modified accordingly. In fact, primitive GLUE operates differently, depending on the feature represented by G_j. If G_j contains only dummy nodes, then the loop of edges bounding each dummy face f becomes an inner loop in the corresponding connection face f' of G_i. If G_j contains dummy arcs and hyperarcs, then each dummy face f' is subtracted from the corresponding connection face

f, thus modifying the exterior loop bounding f and deleting f' as a consequence. This is the case, for instance, for the two features depicted in Figure 2. The following procedures and functions are used as primitives in the description of algorithm GLUE:

—CORRESPONDING_DUMMY_NODE (C_j, D_j, q_j, f): returns the dummy node f' belonging to D_j, which corresponds to the connection node f; that is, $f' = q_j(f)$.

—IS_A_SIMPLE_NODE (G_j, f): returns the value TRUE if f is a simple node in G_j, and the value FALSE if f is a macroconnectionnode.

—ARCLOOP_INCIDENT_TO (G_j, f'): returns the loop l of arcs incident to the dummy node f' of G_j; l will describe the loop of edges bounding the corresponding face in the feature.

—INSERT_INNERLOOP (G_i, f, l): adds the arcs belonging to loop l to the set of arcs of G_i incident to node f, thus defining an inner loop on face f.

—DELETE_DUMMY_NODE (G_j, f'): eliminates dummy node f' from G_j, together with all the arcs and hyperarcs of G_j incident to it.

—SUBTRACT (G_i, f_1, G_j, f_2): modifies the set of the arcs of G_i incident to node f_1 by adding arcs of G_j incident to f_2 as a consequence of the subtraction of face f_2 from face f_1.

—DELETE_DUMMY_MACRONODE (G_j, f'): eliminates dummy macronode f' from G_j by deleting each dummy node forming f' and the dummy arcs and hyperarcs joining simple nodes in f'.

```
Algorithm GLUE (G_i, G_j, C_j, D_j, q_j);
// G_i and G_j are two components of the HFAH such that G_j is a direct descendant of
      G_i in g*;
   C_j is the set of the connection nodes of G_j in G_i;
   D_j is the set of the dummy nodes of G_j;
   q_j is the mapping from C_j to D_j which defines the relation between G_i and G_j //
  for every f in C_j do
    f' := CORRESPONDING_DUMMY_NODE (C_j, D_j, q_j, f);
    if IS_A_SIMPLE_NODE (G_j, f) then
      l := ARCLOOP_INCIDENT_TO (G_j, f');
      INSERT_INNERLOOP (G_i, f, l);
      DELETE_DUMMY_NODE (G_j, f')
    else  // f is a macroconnection node //
      for every simple node f_1 in f do
        f_2 := CORRESPONDING_DUMMY_NODE (C_j, D_j, q_j, f_1);
        SUBTRACT (G_i, f_1, G_j, f_2)
      end for;
      DELETE_DUMMY_MACRONODE (G_j, f')
  end for
end GLUE.
```

The worst-case time complexity of algorithm REFINE is equal to $O(D_jE_j + DD_j)$, where D_j denotes the number of dummy nodes of G_j, E_j the total number of edges on the exterior loops of the faces represented by the dummy nodes of G_j and by their corresponding connection nodes of G_k, and DD_j the number of direct descendants of component G_j in g^*. If components with only dummy nodes are

considered, the time complexity of algorithm REFINE reduces to $O(D_j + DD_j)$ in the worst case.

4.2 Abstraction

An abstraction transformation has a dual effect on the structure with respect to a refinement. The effect of an abstraction applied to an object S represented by a HFAH g^* consists of transforming a part of the boundary of a feature F_i represented by a component G_i of g^*, which is described by a subgraph G_k of G_i, into a new feature F_k and inserting it at a lower level of specification in the hierarchy.

An abstraction applied to a subgraph G_k of G_i consists of splitting G_i into two graphs, obtained by suitably completing G_k and its complement $G_i - G_k$, and of inserting G_k into g^* as a direct descendant of G_i. Subgraph G_k must satisfy the following reducibility conditions:

(i) G_k must be a connected subgraph of G_i.

(ii) If G_i contains a set C of connection nodes of some component G_j ($\neq G_k$), then either $C \subset N_k$ or $C \subset N_i - N_k$, where $G_i = (N_i, A_i)$ and $G_k = (N_k, A_k)$.

In terms of the object represented by g^*, the first condition means that G_k defines a feature in G_i, whereas the second is imposed only in order to maintain the strictly hierarchical organization of g^*. The graph obtained by completion of $G_i - G_k$ is called the *reduced graph* defined by G_i and G_k and denoted $G_i' = (N_i', A_i')$. The set of the connection nodes of G_k in G_i is defined as $C_k = \{f \text{ in } N_i \mid \text{there exist an arc in } A_i - A_k \text{ and an arc in } A_k \text{ incident to } f\}$. In fact, a connection node of G_k in G_i is any node of G_i having arcs incident to it that belong to both G_i and G_k. Thus $N_i' = N_i - N_k \cup C_k$, whereas A_i' is obtained by a modification of $A_i - A_k$. In fact, a transformation could be applied to arcs of $A_i - A_k$ so that G_i' describes an admissible solid object. $A_i' = A_i - A_k$ when G_k describes a feature that defines only inner loops on the object faces. G_k is completed by the insertion of the dummy nodes D_k corresponding to the connection nodes that belong to G_k and, possibly, by the insertion of arcs and hyperarcs that transform the feature into an admissible object. The resulting graph is called *completion* of G_k and is denoted $G_k^* = (N_k^*, A_k^*)$. The tree T describing g^* is modified, because of an abstraction, by inserting G_k into g^* as a child of G_i and transforming all direct descendants of G_i associated with connection nodes belonging to G_k into children of G_k.

If an abstraction transformation is applied to the HFAH depicted in Figure 4c, which splits graph G_0 into two subgraphs representing the main shape and the depression on face f_1 of the object S in Figure 4a, we obtain the HFAH representation of S depicted in Figure 4b. Algorithm ABSTRACT given below is a description of the effect of an abstraction transformation applied to a subgraph G_k of a component G_i of a HFAH g^*. In the description of algorithm ABSTRACT, some functions and procedures defined in Section 4.1 are used as primitives together with the following:

—ADD_COMPONENT_HFAH (g^*, G_k): inserts a new component G_k into g^*.

—CONNECTION_NODES (G_i, G_j): returns the set of the connection nodes of G_j in G_i (G_j must be a direct descendant of G_i).

—DELETE_ARC_HFAH (g^*, (G_i, G_j)): deletes arc (G_i, G_j) from the tree describing g^*.

Algorithm ABSTRACT (g^*, G_i, G_k, C_k, D_k, q_k);
// g^* is the HFAH description of the object;
G_i is a component of g^*;
G_k is a subgraph of G_i describing a feature;
C_k is the given set of the connection nodes of G_k in G_i;
D_k is the resulting set of the dummy nodes of G_k;
q_k is the mapping from C_k to D_k which defines the relation between G_i and G_k //

1: **let** L_k be the set of the loops of arcs incident to the nodes of C_k;
SPLIT (G_i, G_k, C_k, L_k, D_k, q_k);

2: ADD_COMPONENT_HFAH (g^*, G_k);
ADD_ARC_HFAH (g^*, (G_i, G_k));
for every component G_j **in** CHILDREN (g^*, G_i) **do**
C_j := CONNECTION_NODES (G_i, G_j);
if $C_j \subset N_k$ **then**
ADD_ARC_HFAH (g^*, (G_k, G_j));
DELETE_ARC_HFAH (g^*, (G_i, G_j))
end if
end for
end ABSTRACT.

Given a component G_i, a subgraph G_k of G_i defining a feature, and the set C_k of the connection nodes of G_k, algorithm SPLIT, described below, constructs the reduced graph, the completion of G_k, and the set of the dummy nodes D_k of G_k and the mapping q_k (by putting D_k equal to C_k and q_k equal to the identity function). Algorithm SPLIT operates differently, depending on the feature described by G_k. If, for instance, G_k describes a protrusion or a depression on a face or a through hole attached to a pair of faces, then every connection node must be simply added to G_k as a dummy node. If G_k describes a protrusion or a depression on an edge e, then the two faces f_1 and f_2 sharing e become dummy nodes in G_k and also a dummy arc corresponding to e is added to G_k. In this case G_i is also modified, since the two arcs joining nodes f_1 and f_2 in G_i are merged into a single arc that describes the edge on which the feature is attached. If G_k describes a set of faces replacing a vertex (as, e.g., in a bevel on a vertex), SPLIT generates a set of dummy arcs that represents the connections among the dummy faces. A similar situation occurs when G_k describes a set of faces replacing an edge and its two extreme vertices. In this case, however, a dummy arc and two dummy hyperarcs, which correspond to the replaced edge and to its two extreme vertices, are added to G_k together with the dummy arcs describing the dummy edges inserted into the feature (see, e.g., the example in Figure 2).

The following procedures and functions are used as primitives in the description of algorithm SPLIT:

—IS_A_LOOP_ON_A_NODE (C_k, l): returns the value TRUE if loop l is formed by arcs all incident to the same connection node, and the value FALSE otherwise.

—CONNECTION_NODE_CONTAINING (C_k, l): returns the connection node in C_k to which all the arcs of loop l are incident.

—ADD_DUMMY_NODE (G_k, D_k, f'): inserts a new dummy node f' into G_k, thus updating the set D_k of its dummy nodes as a consequence.

—ADD_ARCLOOP (G_k, f, f', l): transforms the arcs of loop l incident to f into arcs incident to the dummy node f'.

—DEFINE_EMPTY_C_MACRONODE (C_k, f): defines an empty macroconnection node f in C_k.

—EXTREME_CONNECTION_NODE (G_i, C_k, e): returns the connection node in C_k, which is an extreme node of arc e of G_i.

—ADD_CONNECTION_NODE_TO_C_MACRONODE (C_k, f_1, f): inserts a new simple connection node f_1 into the macroconnection node f.

—DEFINE_EMPTY_D_MACRONODE (D_k, f'): creates a new empty dummy macronode f' in D_k.

—ARCCHAIN (G_i, l, f_1): returns the chain l' of the arcs belonging to loop l that are incident to the simple connection node f_1; l' will correspond to the chain of edges defined by the feature described by G_k on face f_1.

—DEFINE_DUMMY_NODE_INCIDENCIES (G_k, f_2, l'): defines the set of arcs and hyperarcs incident to dummy node f_2 by completing the set of the arcs in l' and the set of the corresponding hyperarcs with those dummy arcs and hyperarcs that are needed to describe the completion of the feature in order to give an admissible solid object [9].

—ADD_DUMMY_NODE_TO_D_MACRONODE (D_k, f_2, f'): adds the simple dummy node f_2 to the dummy macronode f' in D_k.

—COMPLETE_CONNECTION_NODE (G_i, f_1, l'): deletes the arcs of G_i forming l' from the set of arcs incident to connection node f_1, and completes the set of arcs and hyperarcs incident to f_1 to form a loop, which will correspond to the new loop of edges bounding face f_2 [9].

—DELETE_ARCLOOP (G_i, f, l): deletes the arcs belonging to loop l from the set of the arcs of G_i incident to connection node f.

```
Algorithm SPLIT (G_i, G_k, C_k, L_k, D_k, q_k);
// G_i is a component of the HFAH;
   G_k is the subgraph of G_i describing a feature;
   C_k is the given set of the connection nodes of G_k in G_i;
   L_k is the set of the loops of arcs incident to the nodes of C_k (each loop l in L_k corresponds
         to a loop of edges defined by the feature described by G_k on the object faces);
   D_k is the resulting set of dummy nodes in G_k;
   q_k is the mapping from C_k to D_k which defines the relation between G_i and G_k //
  for every loop l in L_k do
    if IS_A_LOOP_ON_A_NODE (C_k, l) then
      f := CONNECTION_NODE_CONTAINING (C_k, l);
      ADD_DUMMY_NODE (G_k, D_k, f');
      ADD_ARCLOOP (G_k, f, f', l);
    else  // loop l is formed by arcs incident to more than one connection node //
      DEFINE_EMPTY_C_MACRONODE (C_k, f);
      for every arc e in l do
        f_1 := EXTREME_CONNECTION_NODE (G_i, C_k, e);
        if (not f_1 in f) then
        ADD_CONNECTION_NODE_TO_C_MACRONODE (C_k, f_1, f)
      end for;
      DEFINE_EMPTY_D_MACRONODE (D_k, f');
```

```
      for every connection node f1 in f do
        l' := ARCCHAIN (Gi, l, f1);
        ADD_DUMMY_NODE (Gk, Dk, f2);
        DEFINE_DUMMY_NODE_INCIDENCIES (Gk, f2, l');
        ADD_DUMMY_NODE_TO_D_MACRONODE (Dk, f2, f');
        qk(f1) := f2;
        COMPLETE_CONNECTION_NODE (Gi, f1, l')
      end for
    end if;
    qk(f) := f';
    DELETE_ARCLOOP (Gi, f, l)
  end for
end SPLIT.
```

Similarly to algorithm REFINE, the worst-case time complexity of algorithm ABSTRACT is $O(D_k\, E_k + \mathrm{DD}_i)$, where the parameters have the same meaning as in Section 4.1. In the case of components containing only dummy nodes, the time complexity reduces to $O(D_k + \mathrm{DD}_i)$ in the worst case.

5. CONCLUDING REMARKS

We have presented a surface-based model for representing solid objects that is based on the adjacency relations between pairs of faces, and can be organized into a hierarchical form. The model has some important theoretical advantages. All of the three primitive topological entities of a solid object are explicitly encoded in the form of a hypergraph, by representing faces as nodes, edges as arcs, and vertices as hyperarcs. Moreover, it provides a complete topological description of the object boundary, since special topological object features, like depressions, protrusions, or through holes, can be extracted from it, being related to the connectivity properties of the face adjacency hypergraph [8].

The HFAH provides a representation of the decomposition of the boundary of an object into a main shape and its shape features. Hence, despite classical boundary relational models, the HFAH is capable of capturing the global shape of the object and decoupling it from its lower level attributes, thus expressing the object as a hierarchy of volumes described by their boundary. The hierarchical model clearly reflects the process of object design, because it allows both an object definition by face specification and an explicit encoding of the intrinsic hierarchy among the main object shape and its lower level features. Furthermore, this hierarchical model can be locally modified by applying the abstraction and refinement transformations defined in Section 4, which allow the updating of the hierarchical structure either during the design or the analysis phase.

The HFAH has also been demonstrated to be a valid model in practical applications. Because of the possibility of both structuring the model at multiple levels and of decomposing it into its k-connected components [11], the HFAH is especially useful in object recognition applications [8, 9]. Moreover, the model has been shown to be useful in object manufacturing for form feature description. In fact, the HFAH has been used to represent the main shape of the object at the first level in the hierarchy and its form features at further levels of specification [9].

REFERENCES

1. ANCONA, M., DE FLORIANI, L., AND DEOGUN, J. S. Path problems in structured graphs. *Comput. J. 29*, 6 (1986), 553–563.
2. ANSALDI, S., DE FLORIANI, L., AND FALCIDIENO, B. Edge–face graph representation of solid objects. In *Proceedings of the Workshop in Computer Vision, Representation and Control* (Annapolis, Md., Apr. 30–May 2). IEEE, Silver Spring, Md., 1984, pp. 164–169.
3. ANSALDI, S., DE FLORIANI, L., AND FALCIDIENO, B. Geometric modeling of solid objects by using a face adjacency graph representation. *Comput. Graph. 19*, 3 (July 1985), 131–139. (SIGGRAPH '85 Conference Proceedings, San Francisco, Calif., July 22–26.)
4. ANSALDI, S., DE FLORIANI, L., AND FALCIDIENO, B. Form feature representation in a structured boundary model. In *Image Analysis and Processing*, V. Cantoni, S. Levialdi, and G. Musso, Eds. Plenum, New York, 1986, pp. 111–120.
5. BAUMGARDT, B. Winged-edge polyhedron representation. Rep. CS-320, Stanford Univ., Stanford, Calif., 1972.
6. BERGE, C. *Graphes et Hypergraphes.* Dunod, Paris, 1977.
7. CARIMATI, M. Data structure and Euler operators of the FAH representation. Tech. Rep. 12/86 PFTM CADME, Politecnico di Milano, Milan, Italy, 1986 (in Italian).
8. DE FLORIANI, L. A graph-based approach to object feature recognition. In *Proceedings of the 3rd ACM Symposium on Computational Geometry* (Waterloo, Ont., June 1987). ACM, New York, pp. 100–109.
9. FALCIDIENO, B., AND GIANNINI, F. Feature extraction and organization into a structured boundary model. In *Proceedings of EUROGRAPHICS '87* (Amsterdam, Sept. 1987). North Holland, Amsterdam, pp. 249–269.
10. HANRANAN, P. M. Creating volume models from edge-vertex graphs. *Comput. Graph. 16*, 3 (July 1982), 77–84. (SIGGRAPH '82 Conference Proceedings, Boston, Mass., July 26–30.)
11. HARARY, F. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.
12. MEAGHER, D. Geometric modeling using octree encoding. *Comput. Graph. Image Process. 19* (1982), 129–147.
13. REQUICHA, A. A. G. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv. 12*, 4 (1980), 437–464.
14. SILVA, C. Alternative definitions of faces in boundary representations of solid objects. Tech. Memo. 36, Production Automation Project, Univ. of Rochester, Rochester, N.Y., 1981.
15. WEILER, K. Edge-based data structures for solid modeling in curved-surface environment. *IEEE Comput. Graph. Appl. 5*, 1 (1985), 21–40.
16. WEILER, K. Topological structures for geometric modeling. Ph.D. dissertation, Rensselaer Polytechnic Institute, Troy, N.Y., Aug. 1986.
17. WILSON, P. W., AND PRATT, M. Requirements for support of form features in a solid modelling system. Tech. Rep., Geometric Modelling Project, Computer Aided Manufacturing International, Arlington, Tex., 1985.
18. WOO, T. C. A combinatorial analysis of boundary data structure schemata. *IEEE Comput. Graph. Appl. 5*, 3 (1985), 19–27.

Received October 1986; revised April 1987; final revision accepted September 1987