A UNIFIHD, VIEW OF SOLID SHAPE MODELING BASED ON CORSISTENCY VERIFICATION

by

CharlM H. Bastaan & Kennsth Fr«is«

DRC-15-13-82

April, 1982

A UNIFIED VIEW OF SOLID SHAPE MODELING BASED ON CONSISTENCY VERIFICATION

by C. M. Eastman
Carnegie-Mellon University,
Pittsburgh, PA. 15213 U.S.A.

and K. Preiss
Ben-Gurion University of the Negev
Beer Sheva, Isreal

February, 1982

ABSTRACT

The potential benefits of using a canonical model for representing the shapes of solid objects has led to the design and implementation of a number of geometric modelers. A variety of approaches have been used in different modelers, with varying capabilities. This paper generalizes and unifies these approaches by defining the semantics of geometric modeling as semantic integrity constraints. Different methods for representing semantic integrity constraints are reviewed and the solid shape modeling systems that use them identified. The result is a unified view of existing solid shape modeling systems that identifies their similarities and differences with regard to semantic integrity.

1 INTRODUCTION

In many fields of technology, practical applications often precede generalizing theories. Mr. Watt built his steam engine before the laws of thermodynamics were formally expressed. In computing sciences, languages was developed* before the logical structure of languages were dearly formulated. While one may produce good solutions to many problems by judgment or intuition alone, a theory serves to both help produce better solutions than before and to provide a conceptual framework within which one can better understand and extend the implications of various choices.

In the past few years, a number of systems have been implemented and papers written about the computer representation of solid shapes. The common objective of these efforts is to organize information inside a computer so that it correctly represents a solid shape and to provide a set of operations that allow modification of the shape, eg. its transformation into another solid shape. The shape information is then available for use in a variety of applications. The techniques for doing this

have generally been given the name *geometric modeling.* These systems and the papers that describe them present alternative approaches which have much, in common, but are distinct in many aspects. Two surveys have attempted to classify and relate the various results [21,(18].

No geometric modeler yet devised is completely general[1]; each is restrictive at least-according to the types of surfaces by which the shape can be bounded. But if shape modifying operations are to be applied iteratively, then the set of shapes allowed as input and generated as output must be consistently defined in order to avoid errors. The mathematical properties of a consistent representation and especially certain manipulation operations have been defined [19],[20] and are generally referred to as the geometric modeling *well-formedness conditions.* These efforts reformulate and refine the conditions for volume enclosing solids defined in topology and homology theory [7], [8].

This mathematical treatment of well-formedness, however, has no direct correspondence in the implementations of geometric modeling systems. That is, there is no means to directly verify that the well-formedness conditions have been correctly implemented, because of the form in which the conditions have been expressed. This paper presents the constituent requirements of geometric modeling, as defined by the well-formedness conditions, in an alternative form that allows them to be related to methods of computer implementation. To do so, we rely on the notion of *semantic integrity constraints.* Semantic integrity constraints (or just integrity constraints) can be used to make assertions about programs or the states of programs, in order that they may be verified {3]. The ADA language, for example, incorporates facilities for defining integrity constraints for this purpose [21]. Integrity constraints are also proposed as a tool in database design, as a weans^ta embed rules regarding the combination of meaningful values allowed within a database [10]. Methods for defining and managing integrity constraints have been incorporated into several database management systems.

Our purpose is similar to that of program verification; we wish to define the properties of a shape that are "guaranteed to exist" in every object which is modeled; these conditions should be selected so as to correspond to properties of real shapes that are to be represented. With the well-formedness conditions defined as constraints, the techniques for applying constraints in a program can be identified and related to the implementations of geometric modelers attempted thus far. This approach allows comparison of various geometric modeling efforts in terms of their treatment of

[1]This goal is assumed to be impossible.

integrity constraints. More generally, however, an abstract space that encompasses all current geometric modelers is thus defined, allowing investigation of unexplored domains within it. In addition, the paper explores the generality of the use of integrity constraints in geometric modeling.

2 WELL-FORMEDNESS CONDITIONS

In this section we present, in an informal manner, the well-formedness conditions that are known to be necessary for shapes which enclose volume.

In geometric modeling, we are interested in the class of shapes that correspond to the shapes of physical objects experienced in the real world. Generally, these objects are solid, though we sometimes also wish to model the shapes of rooms and other *spaces* that have functions associated with them. In these shapes, there is no ambiguity as to what points in space are inside or outside of the shape. In other words, the shape partitions space into two disjoint domains, one inside and the other outside, one bounded, the other infinite. In addition, the domains defined by a shape are well-behaved, in the sense that each point is completely surrounded by material [19]. This requires that all dimensions of the shape be finite, so that the spatial domain of interest is always the bounded (or inside) one. One last concern is that we are usually interested in shapes that are connected, eg. there is a path within the bounded domain between any two points inside the shape.

2.1 DEFINITIONS

(figure 1 about here)

Space is considered in its usual euclidian form as an infinite point set, contiguous in the three dimensions of X, Y, Z.

A point is a location in space defined by three values for [X, Y, Z].

A surface is a set of points contiguous in two (not necessarily Euclidean) dimensions. An edge is a set of points contiguous in one (not necessarily Euclidean) dimension. A vertex is a bounding point of an edge[2]. The critical property of a surface is that it may be completely triangulated, by which is meant that it may be partitioned into multiple areas, each bounded by three edges and the three vertices that are the ends of the edges. Around any vertex are an adjacent set of triangles for which

[2] Surfaces are sometimes called simplical complexes of dimension two and edges as simplical complexes of dimension one.

the edges opposite the vertex are called the link of the vertex. A surface satisfies the property that the link around any vertex in a surface is either a polygon or polygonal arc[3] [1].

2.2 WELL-FORMEDNESS

One requirement of a solid shape is that its bounding surface be closed. A closed surface is one that can be triangulated so that

1. any two triangles are either:
   - disjoint,
   - have a vertex in common
   - or have two vertices and their common edge in common.

   That is, they do not intersect except at vertices or edges. Also, all edges are simple in the sense that they have exactly two incident vertices.
2. all edges within a triangulated surface are connected; that is, there is a path of edges connecting any two of them
3. for every vertex on the surface, its link is a simple closed polygon. This results in every edge of a triangle being adjacent to exactly two triangles.

We call these the trianguiation, connectedness and link criteria respectively. They are all parts of the more general closed surface criterion [8].

There are many forms of closed surfaces, only some of which enclose volume. A second necessary property required to ensure that the surface encloses a volume is that it is orientable. A closed surface is orientable if its triangles (resulting from an arbitrary trianguiation) can have their vertices ordered consistently, eg. clockwise or counterclockwise, so that the vertex pair of each edge is ordered twice, once from each vertex. This orientability condition is known as *Mobius[9] Law* [14]. An orientable surface partitions space into exactly two regions, one bounded and the other unbounded. The orientability criterion allows unambiguous distinguishing of these two regions. Of course, the bounded region is the enclosed one.

In order to not limit ourselves to shapes whose surfaces are comprised only of triangles, we need to generalize the above family of shapes. Some subset of triangles, that pairwise share common edges,

[3] **A polygon** is **a circuit of** pKnG segments $A_t A_2$. $A_? A_3, \ldots, A_p Aj$ joining consecutive pairs of p points. A polygonal **arc** is a connected subset of tine segments comprising a polygon.

can be combined. This combination is called a face. In usual practice a face is chosen to be that subset of edge-sharing triangles that lie on a surface defined by a single equation or set of parametric equations. All triangles belong to exactly one face. When triangulation is relaxed, then the connected part of the closed criterion ((2) above) must be reformulated, as faces may wholely bound other faces (see Figure 3). The connected criterion may be reformulated as: there is a path from one face to any other face by crossing shared edges[4].

2.3 CONSTRAINT DEFINITIONS

The above criteria can be described in the form of constraints. Those below are only one of several different sets that are equivalent to these criteria.

1. Faces can intersect only at shared vertices or edges (derived from the triangulation requirement).
2. A path must exist between any two faces on a shape, where the path is defined by crossing edges shared by two faces (derived from the connectedness requirement).
3. The link of every vertex is a polygon (derived from the link requirement).
4. When the edges of each face are oriented consistently, eg. clockwise **about a face, each** vertex pair comprising an edge is ordered twice, once in each direction (derived from the orientability requirement).
5. No dimension **or measurement of the shape may be infinite.**

In the modeling of a shape, it is useful to distinguish the topology of a shape from its geometry. The topology defines the-connectedness and structure of the boundary of a solid shape, as defined by its **constituent dements: faces, edges and vertices; the geometry defines the relative placement of the boundary elements in some coordinate system.** *

Of the above constraints, 2, 3 **and 4** are strictly topological. That is, they can be evaluated **by** examining the topology of the shape model, if it is represented explicitly, without recourse to examining its geometry. Constraint 1 requires evaluation of both the topology and geometry. Constraint 5 is strictly geometrical.

---

[4] **This characterization docs not deal with shapes which have enclosed hollows, such as a pressure vessel.**

3 REPRESENTATIONAL METHODS

Baer, Eastman et al [2], Requicha [18] and Wesley [23] present reviews of current methods for geometric modeling. These papers mention at least six methods of representing the shape of a three-dimensional body in a computer:

[a] **Primitive instancing** in which shapes are limited to families of similar shape. Each family usually has a fixed topology and varies according to the relative placement of faces, edges and vertices. Each shape is defined by a fixed list of parameters. For instance, the string [CYLIN, X_{Qf} Y_{Qf} ZQ, X_r Y_r Z_r R] can represent a right circular cylinder arbitrarily oriented in space, with the center line extending from <X_{Qt} Y_Q, ZQ> to <X_r Y_r Z^ and a radius R. The string ['CONE', X_Q, Y_Q, Z_{Ql} X,, Y_r Z_r R] will, in a similar way, represent a right circular cone with base centre at <X_{Qf} Y_Q, Z_Q> and vertex at <X_r Y_v Z^. Note that if <X_r Y_v Z,> = <X_2, Y_2, *ZJ* then the format will not define a 3-D shape. At least two primitive instances have been defined with a variable topology. The first is the *extrusion,* an N-sided base and similar N-sided top, the base and top connected with N rectangles. The second is the *pyramid,* consisting of an N-sided base and vertex top connected by N triangles. In primitive instancing, the name of the shape specifies the topology or class of topologies created. The variable **topologies are** specified by an additional single cardinal **parameter. The dimensions** i.e. **the** geometry, are specified as parameters **also.**

<figure 2 **about here>**

Primitive instancing relies on explicit definitions of the topology or class of topologies created but these definitions may vary from implementation to Implementation. For example, a right cylinder in one implementation may have only one face for its barrel (two opposite edges of the face are joined) and **in another implementation it may have multiple faces (for example, that break the barrel into** quadrants). See Figure 2. Similarly, primitive instancing explicitly defines the dimensions for the geometry of the shape being created. The definition should include the domain of dimensions, or subrange, allowed (eg. are negative dimensions allowed?). Some dimensions or combination of dimensions for a primitive instance may lead to violation of the triangulation or orientability requirements. Some languages permit the definition of data types with subranges.

In this method of representation, all well-formedness conditions are embedded in the procedures that operate on the shape definition specified. Several different means may be used to guarantee that the well-formedness conditions are met. These will be discussed in Section 4.1 and 4.2.

(b) Spatial occupancy enumeration is another representation method. In it, space is partitioned into a tessellation of cells. Each cell is specified by an index; there is usually a simple mapping from the index to the spatial location of the cell[5]. A shape is a set of cells. This representation can specify any general shape, provided the element-size is small enough to give good resolution.

Interpretation of spatial occupancy enumeration consists of logically combining cells of similar occupancy. Spatial occupancy enumeration imposes its own topology on all shapes being described. It is difficult to define rules for consistently mapping into this topology and interpreting it. For example, after representing a polyhedron with a skewed face, how should its bounding edges and vertices be located? Within the topology imposed, constraint 1 is guaranteed by the the structure used; 3 and 4 are managed by the procedures interpreting the representation. Constraints 2 and 5 must be guaranteed by the procedure writing into the representation.

Because of the difficulty in consistently interpreting the surface topology in spatial occupancy enumeration, it is not suitable for graphic interpretation; it may be useful for calculating spatial pr&perties and for spatial conflict testing.

(c) Octree representation in which a cubic volume enclosing the shape is split into eight octants [12]. Each octant if completely full or empty, is so marked and is then a terminal node on a tree, else the octant is split into eight and so on recursively until a full or empty octant is reached, or until a cell of size equal to the resolution of the model is reached.

This method is similar to spatial occupancy enumeration, but uses less memory due to the aggregation of cells together. Constraints 1 and 5 are managed as for spatial occupancy enumeration. This method can be used for graphic display [4J.

(d) Cell decomposition, which is a more general case of (b) above, in that different primitive shapes and with varying sizes can be specified. In general, two primitive shapes can be related in one of five relations; they are *disjoint* if they have no points in common; they are *subjoint* if all the points in one shape are common with the other shape (but not vice versa); they are *tangent* if the shapes only share points along their boundary; thay are *conjoint* if they share some points but not all, and they are *coincident* if all points in both shapes are common with the other. In cell decomposition, primitive shapes may be tangent or subjoint. A cell is defined as a type with its dimensions and locations,

---

[5] for example, mapping first into a three-dimensional array, then scaling from the array into the coordinates of the proposed shape.

represented by some method such as (a) above. An arbitrary shape can be represented only if it can be made by "sticking together" or "cutting out" cells of the given collection of possible shapes. The greater the number of shapes, the more general the representation.

Interpretation of a model based on cell decomposition involves aggregating the properties of the primitive shapes. Similarly to spatial occupancy enumeration, the topological well-formedness constraints are embedded in these procedures. The triangulation requirement is guaranteed by the shape definition procedures that prohibit more than one solid or more than one void from jointly occupying points not in their bounding surfaces.

<figure 3 about here>

(e) Constructive solid geometry, (CSG) can be thought of as a generalization of cell decomposition in that in addition to tangent and subjoint relations between two primitives, conjoint relations are also allowed. This generalization allows definition of the spatial set operations of negation, union, intersection and differencing. An example is given in Figure 3. The geometric feasibility of the shape is ensured by guaranteeing valid representations for each primitive solid, and by ensuring that the set operations do not introduce infeasible faces or other artifacts. Note that each primitive solid has to be represented somehow, as in (a) above. Constraint 1 is guaranteed by the structure of the primitive solids and by the operators that derive the locations of new edges and vertices resulting from the combining of shapes. Constraints 2,3,4 and 5 also must be guaranteed by the manner of derivation of the primitive solids and by the operators that combine them.

(f) Boundary representation is a method in which sufficient information about the surface boundaries is given to construct the shape. There are various collections of boundary information that may be used, the only requirement being that the information be sufficient to define all bodies in the class represented. Suitable representations are:

1. face and edge based structure:

   - 3 space coordinates of each vertex
   - a double <vertex 1, vertex 2> specifying which vertices are connected by an edge (and possibly the parameters defining the space surve of the edge)
   - an n-tuple specifying which n edges form a loop
   - an m-tuple specifying which m loops form a face and the parametric representation of the face surface; this is demonstrated in Figure 4 for the box of Figure 1

<figure 4 about here>

2. loop based structure:

- 3 space coordinates of each vertex
- an n-tuple specifying which n vertices form a loop
- an m-tuple specifying which m loops form a face and the parameters that define the surface of the face

3. edge based structure (for planar faces):

- 3 space coordinates of each vertex
- for each edge, a list of 6 vertices. There are the 2 vertices at the ends of the edge, and another 4 "nearest neighbour" edges found by tracing along the two faces which meet at that edge. This is known as the "winged-edge" representation, and is demonstrated in Figure 5 for the box shown in Figure 1.

<figure 5 about here>

Other combinations of information are also possible.

In boundary representations, the data structures can represent the topology isomorphicalty; that is, there can be a one-to-one correspondence between the data structure and each face, edge and vertex. In the isomorphic representations, the operators that create or modify the data structure are responsible for the topotogicaJ constraints (in contrast to the other representations, in which the constraints must be embedded in the operations that read the internal data). The triangulation constraint requires that a check be made that no faces intersect otherthan pcarwiseatong edges or at vertices. In the non-tsomorphic boundary representations, the interpreting procedures must evaluate the topologtcal constraints.

Boundary representations may be initially defined by a user in various ways in addition to those above, such as (a), (e) or (g). In addition, a special set of operators, called the Euler operators, have been developed that allow definition of a topology or combined topology and geometry incrementally [6]. These operators allow for the partitioning the face of a sphere topology[6] or for gluing together matching faces of a shape to make new shapes of higher genus. The Euler operators can guarantee conditions 2,3 and 4 by proper management of the data structures involved. Conditions 1 and 5 must be evaluated by checking the geometry with the topology.

---

[6] A sphere topology is made up of a single face which can be thought of as being shaped by a baMoon face so as to be bounded by a single vertex.

(g) Half space equation representation, in which each face is represented by the parameters of its equation and an orientation. The equations are the equations of a plane, or cylinder, or cone, or spline surface, etc., as required. The locations of vertices and definitions of edges are computed from the intersections of the surfaces. -Because various combinations of intersections are possible (for non-convex shapes), a sequential ordering of surface combinations is required. This ordering allows derivation of the topology from constraint 1. Like CSG. the resulting topology must conform to constraints 2, 3 and 4. Constraint 5 is guaranteed by the constructing operations and also the primitive surface types.

3.1 THE RELATION OF METHODS TO GEOMETRIC MODELING SYSTEMS

Geometric modelers with any degree of general capability have the ability to define primitive elements (shapes, surfaces or topological entities) and to compose them. This usually requires a combination of methods. For example, the GLIDE geometric modeling package relies on methods (e) and (f) [6],[13]. PADL relies on methods (a), (e) and (f) [22]. Synthavision seems to rely on methods (a) and (e) [9].

Correspondingly, the well-formedness constraints are managed in each geometric modeler by several methods. These methods must be verified to be both complete and consistent in combination. The structures used to manage the integration of methods are reviewed in Sec. 6.

4 IMPLEMENTATION OF CONSTRAINTS

While constraints may characterize conditions of many sorts, their treatmemkrgeometeria modeling systems provides a means of classifying geometric modelers.

Different geometric modeling methods impose the well-formedness and shape family constraints in different ways. Usually these are imposed in a manner that is implicitly defined by the representation method used.

4.1 TOPOLOGICAL CONSTRAINTS

Modeling efforts of any type may impose integrity constraints in either of two phases of the modeling process: when *defining* the model (writing it) or when *interpreting* the data (reading it). There are no other options.

There is only one means known to the authors to enforce topological constraints during definition

of a shape model. If the data structure has an isomorphic mapping to the topology of a shape, then the operations defining the model can enforce the topological constraints. Isomorphism must exist regarding the properties of the topology of a well-formed shape with properties of the model data structure. For example, the topological construction operations can guarantee that the link of a vertex is a polygon. Because of the needed property of isomorphism, only the boundary representation is capable of evaluating topological constraints while the model is being defined.

If isomorphism does not exist, then the operators that interpret the shape model must do so in a manner that satisfies topological constraints. Most representations involve many primitive shapes, each with their own explicit or implicit topology. Interpretation of the modeled shape involves the aggregation of the individual shape models. If the allowed combinations are only along shared faces, (this is a usual restriction that is both imposed by the representation and also consistent with the need to create vertex structures with polygon links), then this relationship defines the topology combining operation required during interpretation.

Most geometric modelers rely on a set of primitive shapes, defined by primitive instancing (method (a)). For each primitive, there is a corresponding simple fixed topology or topology class. The fixed topologies are usually pre-defined in order to reduce the computation required when instancing the primitive. (Some geometric modelers do not allow creation of a new shape primitive topology.) Those primitives that rely on a topology class, such as the extrusion, must compute their topologies as they are needed. But as hew instances of shapes are created, a check can be made to see if that class of topology has already been constructed, in which case it can be used again. For example, in some GLIDE applications, a table is kept of extrusion topologies made. If a desired topology structure matches an existing one, it will be used again rather than recomputing it

The half-space representation combines surfaces sequentially so as to result in a well-defined topology. This combining operation must guarantee that the resulting information is consistent with topological constraints.

It should be noted that many properties of a shape model are not affected by the topological wen-formedness constraints. Areas and mass properties can be derived from many shape models without certain topological considerations (and thus without requiring complete well-formedness).

4.2 GEOMETRICAL CONSTRAINTS

Different methods may be used for implementing geometrical constraints. Constraint 5 is usually imposed through the definition of primitives or surfaces used in defining the shape. Explicit testing is not required. For constraint 1, a check must be made that the values defining the geometry are consistent with the topology. In those modelers where the topology is computed from geometrical information, then this consistency is guaranteed by the derivation process. In the boundary representation, which represents the topology, consistency of the geometry and topology must be explicitly checked.

4.2.1 CONSTRAINTS ON SHAPE CLASSES

The geometry of a shape is restricted not only to be well-formed, but also to depict a certain class of shapes. The reason for constraining the shapes to such a class is in order to: (1) simplify their input specification so as to require only the minimal or non-redundant specifications for the shape class, and (2) to simplify the construction of the shape models. As an example, a right cylinder cannot include shapes with unequal radii along the axis. This reduces the parameters required to define the shape and associates the shape defining process with appropriate mathematical techniques (such as conics). In this section, we identify different means by which the set of possible shapes may be constrained to subsets of the well-formed set.

<figure 6 about here>

We rely on the boundary representation in the development of this example. The data structure shown in Figure 4, as it stands, will represent any hexahedron, whether well-formed or not. The set of shapes to be characterized can be defined by applying constraints to the three nested sets shown in Figure 6. These sets are noted by super scripts, and correspond to:

1. vertices on the same face
2. opposite faces
3. a closed body

If we consider the class of hexahedral shapes, in a scale from less constrained to more constrained, then we can order them as follows, each with the set of constraints that defines them:

(a) a self-intersecting (impossible) shape: (no constraints)

(b) a non-self-intersecting hexahedron: (no points exist within one face that also falls within another face, except along edges or vertices (constraint 1)).

(c) a plane hexadron, without self-intersecting faces: (within each of .the 6 face sets, unique parameter values of a, b, c and d exist to satisfy the equation

$$ax + by + cz \bullet d = 0$$

(d) a plane hexahedron with parallel faces: (add the constraint that for each of the three sets of opposite faces, the radio of the parameter a from one face to the parameter a in the other face i$ equal to the ratios of b and c for the two faces.)

(e) a box with all right angle corners: (note that any one vertex is included in just three sets. If we call the set of those 3 sets the intersecting set, 8 intersecting sets exist. For each of those intersecting sets, add the constraint that

$$^{a}i^{a}i * {}^{b}i^{b}j \bullet {}^{c}i^{c}j = \circ$$

for the cases: i = 1, j = 2; i = 2, j = 3; i = 3, j = 1.)

(f) a cube with allskies having the same dimensions: (add the constraint that for each of the three disjoint pairs of the six sets, the differences in $d/te,^{2} \bullet bf \bullet c;^{2})^{1/2}$ are equal.)

Other representations than the nested sets in Figure 4 could be used to express the above set of constraints. However, the sets of constraints would have the properties of being additive and would demonstrate the same overall point: that the constraints defining classes of shapes are extensions to the constraints required by geometric well-formedness-

4.2.2 METHODS FOR CONSTRAINING SHAPE GEOMETRY

Methods for imposing these constraints upon input include:

1. restrict the degrees of freedom allowed on the geometrical dimensioning of the shape class. In our example of the constraint set which sequentially restricts the allowed shapes of an hexahedron, the degrees of freedom are:

   - (a) = infinite (three for each surface point)
   - (b) » infinite (same except for pair wise inequality constraint)
   - (c) « 18 (6 sets of face coefficients, the d values being redundant)
   - (d) = 9 (one coefficient and two ratios, each for 3 face sets)
   - (e) = 3 (3 distances)
   - (f) = 1 (1 distance)

Both (a) and (b) do not incorporate any representation scheme for defining the surface points (as in (c)), thus allowing each surface point to be defined independently. These restrictions on the degrees of freedom may be implemented as procedures triggered when coordinate values are assigned to vertices in Figure 4, or directly into the data structure by limiting the cardinality of values stored and using pointers to the common values [5] and/or by restricting the number of parameters the user may specify. This is shown in Figure 7.

2. In addition to restricting the degrees of freedom, it is also sometimes necessary to rely on a process that checks values on input and only assigns them if they satisfy constraints. This means of imposing constraints is managed by the parsing process and has been used in database management systems as a means to impose semantic integrity constraints [10].

3. allow any assignment, then test if the assignment is correct when evaluating it. This is essentially the process of generate-and-test, a weak problem-solving method in that the computational requirements are greater than necessary.

The first two methods of constraint imposition on the geometry apply during the definition of a shape model; the last one applies during interpretation.

The means of implementing geometric constraints in a geometric modeler are largely hidden from a user, they are embedded in the coding. It appears that most rely on restrictions on the degrees of freedom to implicitly define classes of shapes. This is clearly the case in primitive instancing. There may be important reasons, however, for making the constraints explicit. They precisely define the class of shapes that must be dealt with by operators on ttie shape. They also specify the conditions to which any combination of input must be restricted (for example, when the thickness of a shape is zero). Thus they serve the purpose of assertions that can be used taverify the code produced [3].

<figure7abouthere>

The capabilities of the seven representation methods to maintan 3-D shape consistency are summarized in Table 1.

5 SOME CONSIDERATION IN SELECTING INTEGRITY MANAGEMENT SCHEMES FOR GEOMETRIC MODELING

In the structures described, well-formedness requires both that the primitive shapes be well-formed and that each of the structuring operations propagate the well-formedness conditions and/or guarantee the well-formedness for any new combination of shapes. This combining process may be done during the input and storing of the shape information (writing) or during its interpretation (reading).

Table 1

Capability of different representation Methods *

to maintain consistency

METHOD	3-D FEASIBILITY GUARANTEED BY		SHAPE GUARANTEED BY	
	Data Structure	Procedures	Data Structure	Procedures
Primitive Instancing	NO	YES (1)	YES	NO
Spatial Occupancy Enumeration	YES (2)	NO	NO	YES
Octtree Encoding	YES (2)	NO	POSSIBLY (3)	YES
Cell Decomposition	NO	YES (4)	NO	YES (4)
CSG	NO	YES (4)	NO	YES (4)
Boundary Representation	YES and/or	YES	YES and/or	YES
Half Space Representation	NO	YES	NO	YES

Notes:
1. The format does not in itself guarantee that the body is feasible, e.g., the vertex of a cone may be in the base plane.
2. Assuming unconnected bodies are allowed.
3. Using a fixed tree structure is possible but it is unlikely in practice.
4. Assuming each primitive body is guaranteeed consistent.

At one extreme is the combining of information only during interpretation. This is the approach followed by Synthevision, for example [9]. Synthevision computes the surface orientations and location and shape properties by a sampling method, passing "rays" through the combination of shape primitives at uniform intervals. Ray passing is done during interpretation and it integrates the separate definitions of the shape primitives.

An intermediate approach relies on some processing of the combined shape information upon input and further processing at interpretation time. This approach is used in TIPS, developed by Okino's group at Hokkaido University [17]. TIPS maps the parametrically defined solids defined in the input into a three-dimensional . The array cells can be pre-processed to determine the proportion filled for deriving mass properties. For drawings or sections, the array is used to access relevent parametrical solids in each region of the final shape. Thus the mapping into the array and the computation of mass properties is done once, probably at entry; the drawing information is generated when it is needed, during interpretation.

At the other end of the range are modelers using the boundary representation. Here, inputs are combined into a structure isomorphic to the topology, in which case the combining operators take full responsibility for the topology well-formedness constraints. The geometric constraints also are checked as the geometry is entered or modified. Thus there is no checking during interpretation.

In CAD, it is usually the goal to use the geometric model to integrate several applications. That is, the information is entered once, then read several times. This ratio of use identifies an advantage to modeling schemes that check well-formedness during definition (as versus during interpretation). The methods that require checking during interpretation can reduce unnecessary checking by imposing an interpretation at various points during design and hence checking the well-formedness of the model once only prior to actual interpretations. If there are no later writes, then later interpretations need not check again. For a geometric modeler supporting continuous design development, this approach still imposes serious overhead, because the scope of checking during interpretation is not matched to the operations of entry; global checks are often required. This suggests that there is an inherent advantage to modeling schemes that check well-formedness constraints when data is entered into the model, such as boundary representations, and hence do not need checking later.

6 COMPLEX SHAPES AND ASSEMBLIES

There is recent interest in extending the concept of well-formedness beyond single shapes, to assemblies. In assemblies, the well-formedness constraints are defined by the functional requirements of the assembly [5,16].. These well-formedness conditions can be approached in a similar manner to well-formedness, but with the addition of a new class of constraints.

In the modeling of single shapes, well-formedness relied on only two types of constraints: topological and geometrical constraints. In assemblies, in addition to these two, there is a third type of constraint that relates the position of the two shapes together, that is, there are *location* constraints [5]. in a complementary manner, topological constraints have only a minor role in well-formedness at the level of assemblies.

<figure 8 about here>

<figure9abouthere>

As an example. Figure 8 shows an assembly made up of two of the shapes of Figure 3 and a plate. Similarly to the representation of shapes, the representation of assemblies relies on a hierarchical structure.

Constructive solid geometry (CSG) represents a shape as a n-level tree. Conceptually, the shape is broken into sub-shapes, then sub-sub-shapes and so on until the level of primitive solids is reached. The partitioning is non-unique (see Figure 9) and corresponds to the reverse of the spatial set operation. The primitive solids at the bottom of the hierarchy are represented by an appropriate structure for evaluating the constraints. Relations between sub-shapes at level H[7] and the feasibility of each level i is preserved by the spatial set operators, which generate the higher level shapes from the lower level ones.

<figure 10 about here>

The representation of assemblies relies on a hierarchical structure similar to the representation of shapes. An example is shown in Figure 10. In an assembly, when a constraint relation is stated at some level in the structure, it gives rise to constraints at the lower levels. The lower level constraints can be thought of as a more detailed relation of the higher level constraint. For instance, for the

[7] The levels in the hierarchy are numbered from the most aggregate to the most detailed.

assembly of Figure 3, if one specifies that the two horizontal holes be collinear, each level below in the hierarchy will have an appropriate constraint Depending on the levels of hierarchy chosen these could include the constraints (see Figure 10):

- At the level of the complex shape, the horizontal holes of B_1 must be collinear with the horizontal holes of B_2
- At the level of a simple shape, the horizontal hole of P, of B_1 must be collinear with the horizontal hole of P_t^* of B_2
- At the level of faces,
- the central normal to the end face of C_t of P_1 of B_1 must be collinear with the central normal to the end lace of (^ of P_1 of B_r or
- At the level of vertices, the center vertices of the two plane faces of C_t of P_x of B_t must be collinear with the center vertices of C_t of P_t of B_2.

In the above example, if the assembly were directly defined as boundaries, in a 1-level hierarchic definition, then only the last two mentioned constraint would be applicable. At each level the constraint is specified so as to apply to the information at that level.

Similar to the definition of well-formedness of shape classes, constraints can define conditions that an assembly must meet Wheras the conditions defining a shape class are to facilitate shape defining, the constraints associated with an assembly are generally defined by its function. Like shape families, these constraints define the conditions that must be respected by operations that manipulate an assembly, unless the constraints are explicitly varied.

7 CONCLUSIONS

We have formulated the well-formedness conditions of geometric modeling as integrity constraints. This allows them to be related to the implementation of geometric modelers and opens the door to verifying the correctness of modelers, a capability not currently possible.

It has also been shown that a useful characteristic of constraint management in geometric modeling is whether the constraints are managed during entry or during interpretation. This distinction is important because of the different ratios of reads to writes in CAD systems. We believe that well-formedness checking at entry is to be preferred for interactive systems.

<figure 11 about here>

Representation of shapes and assemblies can be viewed as a multi- level hierarchic data structure, with integrity constraints imposed at various levels. Certain constraints apply at particular levels, depending on the representation. The whole structure can be conceptually seen as being in three different general classes (Figure 11). The upper levels contain data relating, from the top down, assemblies to sub-assemblies, eventually sub-sub .... sub-assemblies to complex shapes, complex shapes to sub-shapes, and sub-sub...sub-shapes to simple shapes. The next class of levels contains topological information, or information about which vertices are related to which edges, loops or faces. Constraint relations in this class of levels define the shape of the object without giving vertex assignments. The lowest class of levels contains full geometric information, in that face, edge or vertex location assignments are made.

The structure can be data-driven, that is, with shape information implicit in the structure, in which case the structure relies on a data structure tsomorphic to the structure of the shape topology. An alternative is a procedure- driven structure, when the shape information is contained in constraint relations that operate (during reading or writing) on different parts of the structure.

8 REFERENCES

1. Alfors, LV. and L Sario, Riemann Surfaces. Princeton University Press, 1960.

2. Baer, A., Eastman C and Henrion M. -Geometric Modelling: A Survey", Computer Aided QeSCUL Vol. 11, No. 5, pp 253 • 272.1979.

3. Dijkstra, E.W. JtOJacJoline of Program mine, Prentice-Hall, 1976.

4. Doctor, L.G. and J.G. Torborg, "Display techniques for octree encoded objects", IEEE Computer Graphics and AftniffltftofM!, 13* July, 1981, pp. 29*38.

5. Eastman, C. M. -The Design of Assemblies ", Institute of Physical Planning, Carnegie-Mellon University, Pittsburgh, PA 15213, Report No. 11,1980.

6. Eastman, C. and K. Weiler, -Geometric modeling using the Euler operators", Proc. Conf. on Computer Graphics in CAD/CAM D. Gossard, ed. MIT Press, 1979.

7. EHenberg, S. and M.E. Steenrod, Foundations gf Algebraic TOPOIOOV, Princeton University Press, 1952.

a Giblin, P.G. Graphs. Surfaces and Topology Chapman and Hall, London, 1977.

9. Goldstein, R. and L. Malin. "3D Modelling with the Synthavision system", Proc. on Computer Graphics |n CAD/CAM D. Gossard, ed. MIT Press, 1979.

10. Hammer, M. and D. MacLeod, "Semantic integrity in a relational database system", Conf. on Very Large Databases, 1975.

11. Jakobsen K. "The Concepts of Topology, Form and Dimension in Computer Description of Mechanical Products", EURO IFIP Conference, North- Holland Publishers, pp 515 - 520, 1979.

12. Jackins, C.I. and S.L. Tanimoto, "Octree and their use in representing three-dimensional objects", Comp. Graphics and Image Processing, 14, pp. 249-270, 1980.

13. Kalay, Y. and C. Eastman, "Shape operations: an algorithm for spatial-set manipulation of solid objects", Technical Report 16, Institute of Building Sciences, Carnegie-Mellon University, July, 1980.

14. Klein, F. Geometry: Elementary Mathematics From and Advanced Standpoint, Dover Publications, 1939.

15. Pratt, T.W. Programming Language: Design and Implementation Prentice-Hall, 1975.

16. Preiss K. "Data Frame Model for the Engineering Design Process", Design Studies, Vol 1, No. 4, 1980.

17. Okino, N. "TIPS-1: technical information processing system for computer aided design, drawing and manufacturing" in Computer Languages for Numerical Control, J. Hatvany, ed. North-Holland, 1973, pp. 141-150.

18. Requicha A. G. "Representations for Rigid Solids: Theory, Methods and Systems", ACM Computing Surveys, Vol. 12, No. 4, Dec. 1980, pp 437 - 464.

19. Requicha, A.A.G., "Mathematical models of rigid solid objects", Tech. Memo. 28, Production Automation Project, University of Rochester, N.Y. 1977.

20. Requicha, A.A.G., and R. Tilove, "Mathematical foundations of constructive solid geometry: general topology of regular closed sets", Tech. Memo. 27, Production Automation Project, University of Rochester, N.Y., March, 1978.

21. SIGPLAN Notices, "Preliminary ADA Reference Manual" SIGPLAN Notices 14:6 June, 1979.

22. Voelker, H. et al, "The PADL 1.0/2 system for defining and displaying solid objects", ACM Comp. Graphics August, 1978, pp. 257-263.

23. Wesley M. Chapters 1 and 2 of Computer Aided Design: Modeling, Systems Engineering, CAD-Systems, Springer Verlag Lecture Notes in Computer Science 89, 1980.

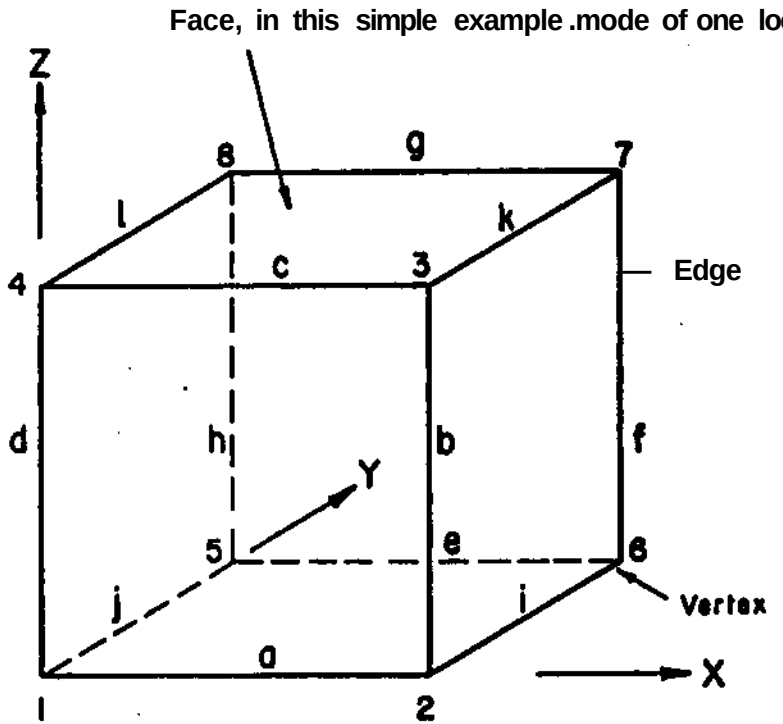


FIG. 1 A SIMPLE BOX SHAPE USED TO ILLUSTRATE THE CONCEPTS.

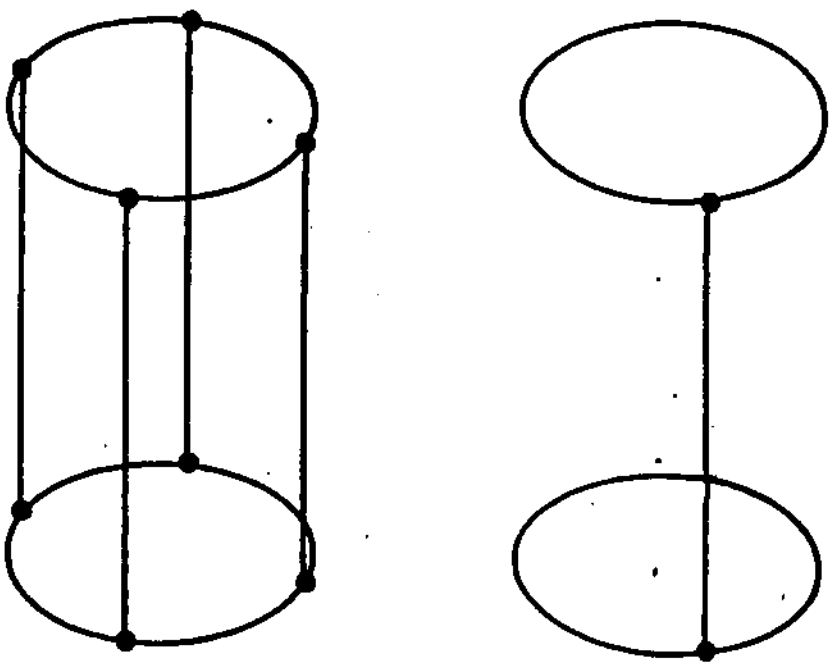


FIG. 2 DIFFERENT TOPOLOGIES FOR THE SAME BODY.

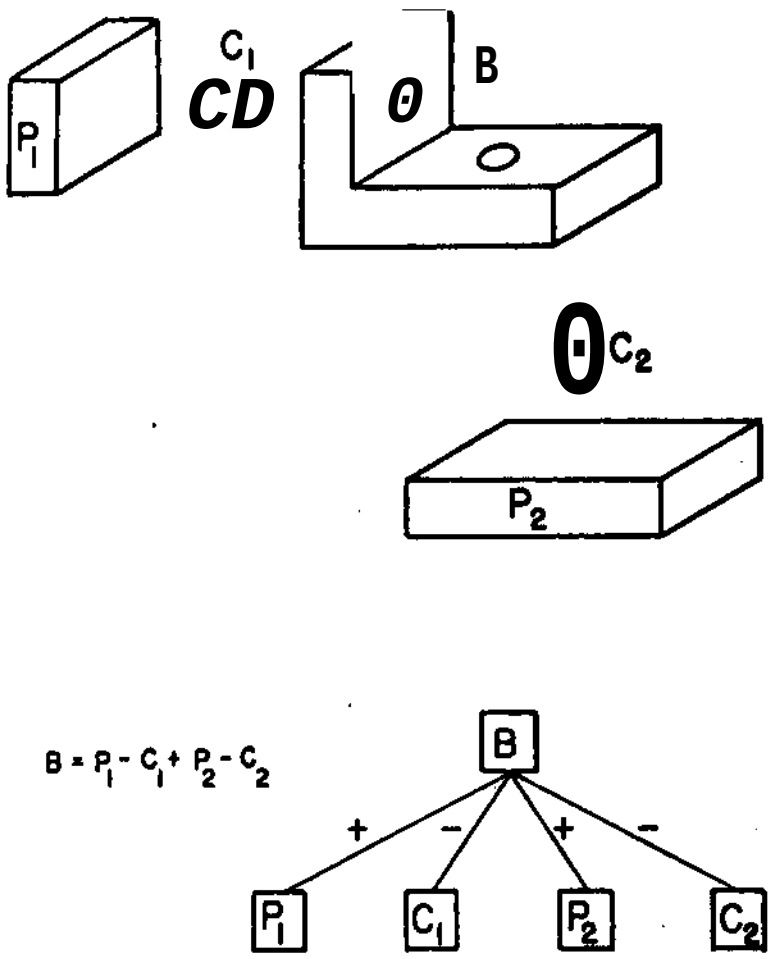


FIG. 3 A COMPLEX BODY MADE UP OF 4 SIMPLE BODIES.

The operators shown are addition and **substraction.** Union, intersection and symmetric difference **are** other possible operators.

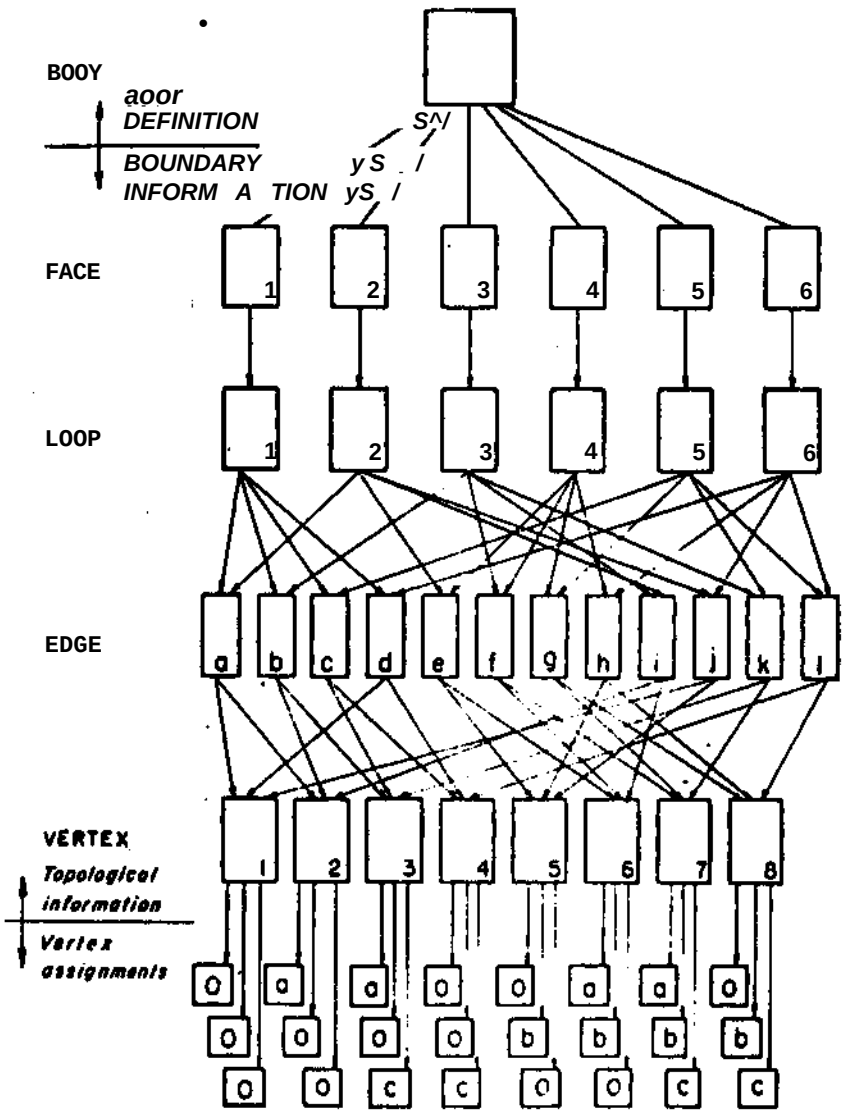
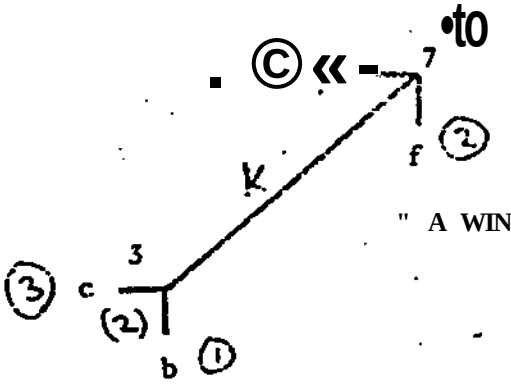


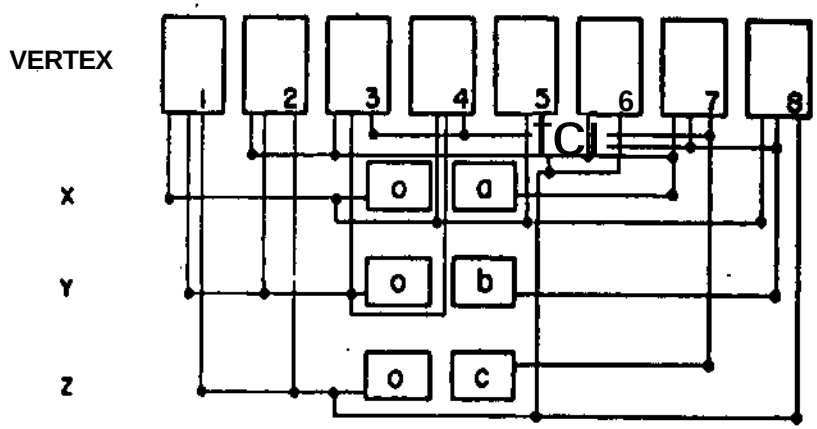
FIG. 4 A BOUNDARY REPRESENTATION OF THE BOX OF FIG. 1.



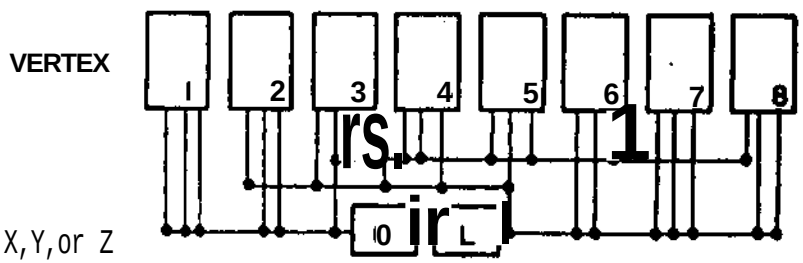
" A WINGiU). CDGli

EDGE INDEX	VERTICES		NEIGHBORING EDGES			
	(1)	(2)	rr>	to	3	4
a	1	2	j	i	d	b
b	2	3	i	k	a	c
c	3	4	k	1	b	d
d'	4	1	1	3	c	a
e	6	5	i	j	f	h
f	7	6	k	i	9	e
a	0	7	1	k	h	f
h	5	8	j	1	e	9
i	2	6	a	1	b	f
j	5	1	e	a;	h	d
k	7	3	9	c	f	b
1	4	8	c	9	d	h

FIG. 5 A UN'CiJO-EDGU JUlfRESLNTATIGN Or" THE HDGL5 ANT; LC TS OF FIG. 1.



Coordinate data structure for a box



Coordinate data structure for a cube

FIG. 7 THE DATA OF FIG. 4 STORED BY INDEXED VALUES OF COORDINATES. THIS METHOD INCORPORATES SHAPE DATA IN THE DEFINITION OF THE DATA STRUCTURE. ASSIGNMENTS ARE IN LOCAL COORDINATES.

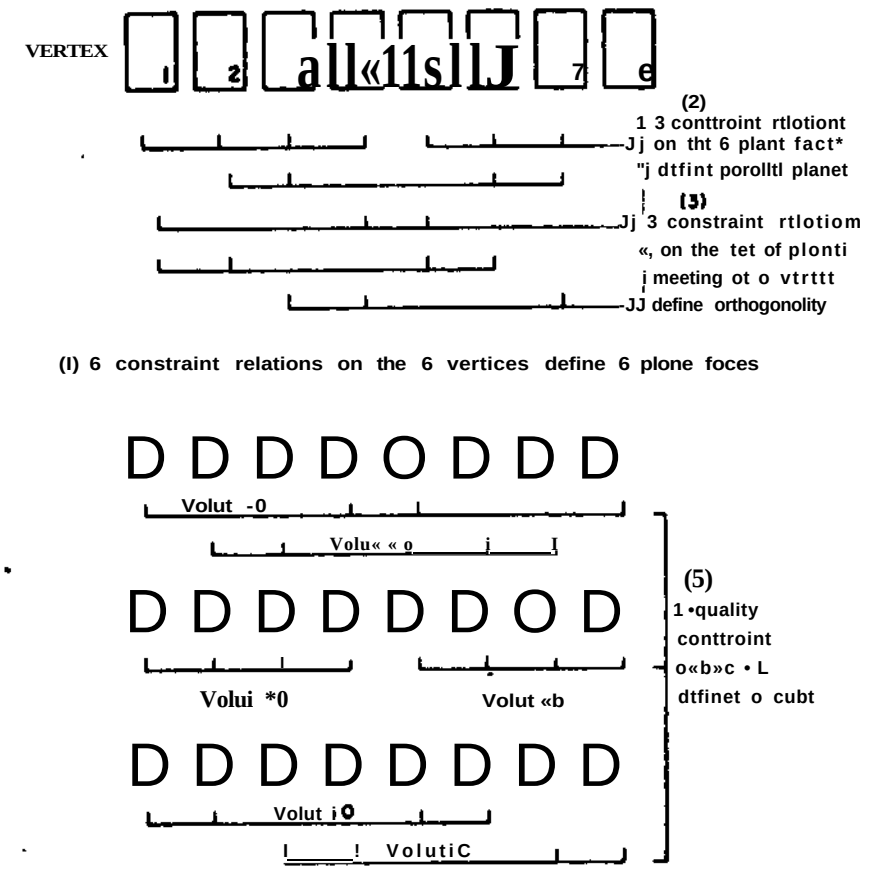
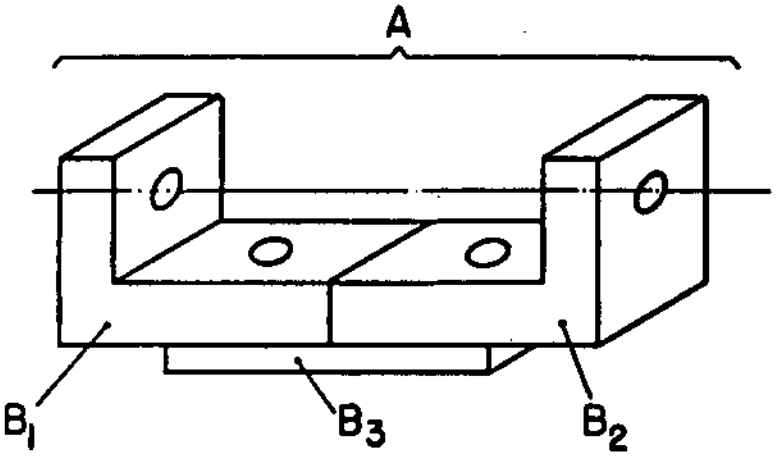


FIG. 6 CONSTRAINT RELATIONS ON THE TOPOLOGY DATA DEFINE SHAPE.



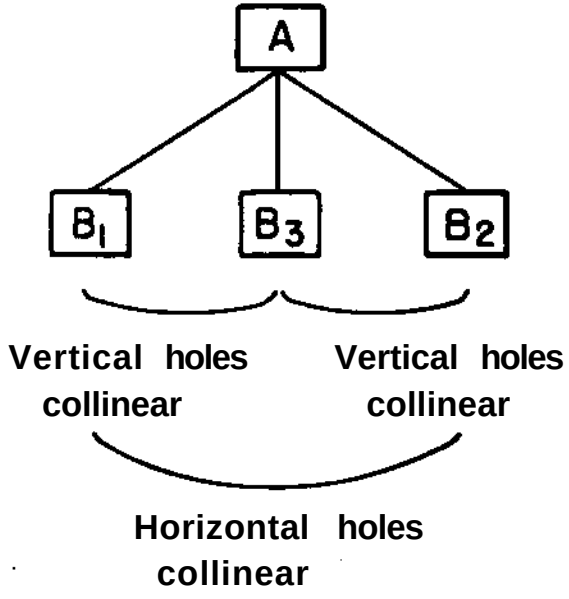


FIG. 8 AN ASSEMBLY AS A HIERARCHY OF BODIES WITH CONSTRAINT RELATIONS BETWEEN THEM....***

$B \bullet (P_1 - C_1) \bullet (P_2 - P_2)$

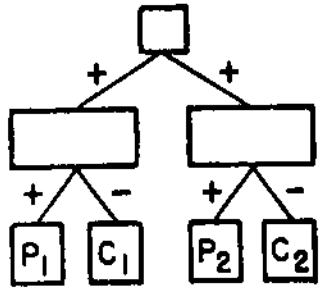


FIG. 9a. AS FIG. 3 BUT IN A 3-LEVEL HIERARCHY.

$B\text{-}(P_1\text{-}(C_1\text{-}(P_2\text{-}C_2)))$

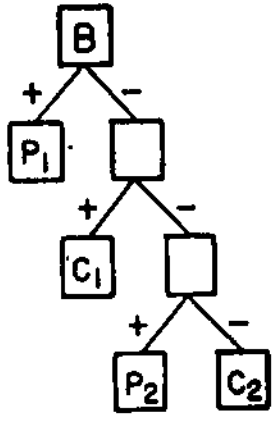


FIG. 9b AS FIG. 3, BUT AS A MULTIPLE LEVEL BINARY TREE.

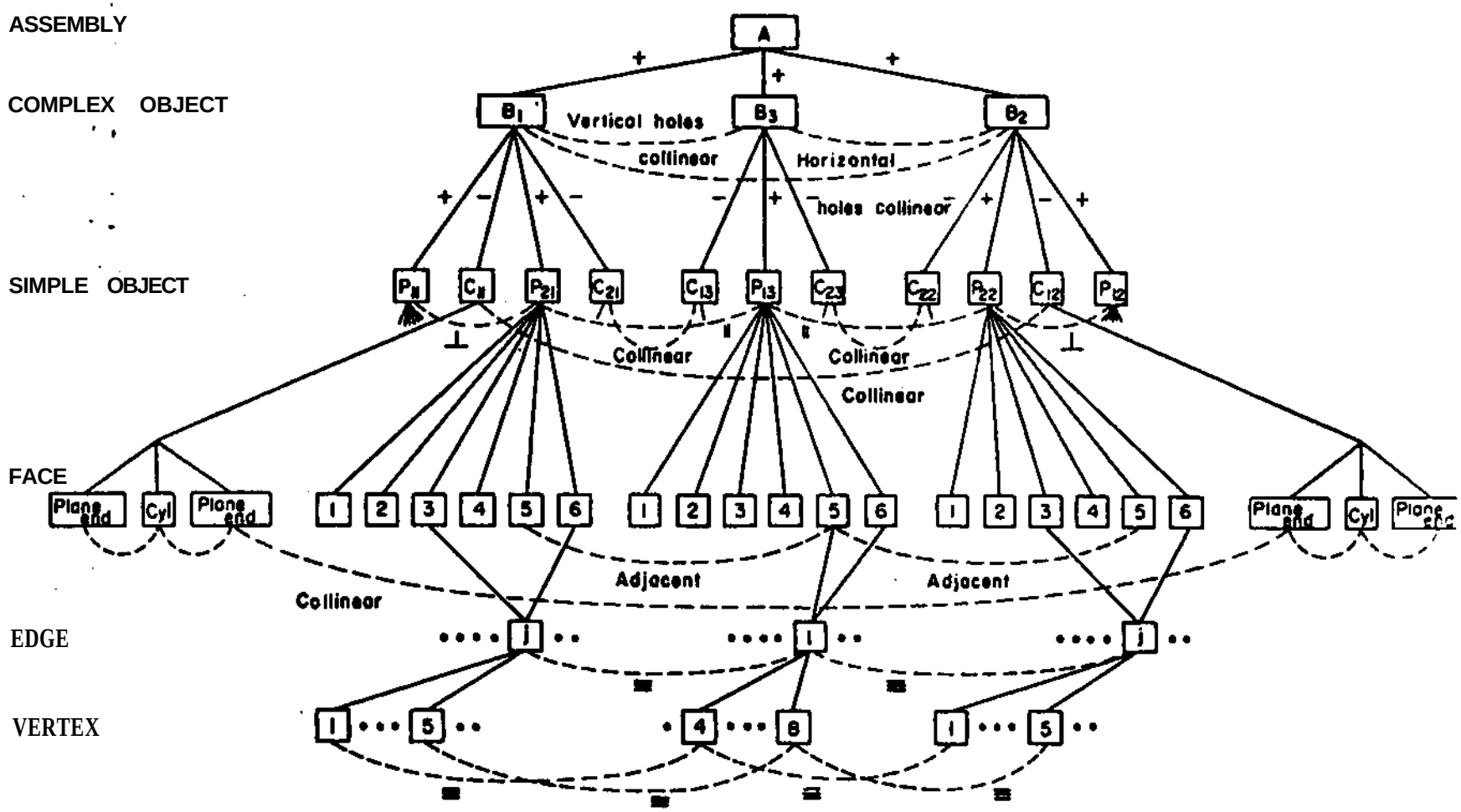
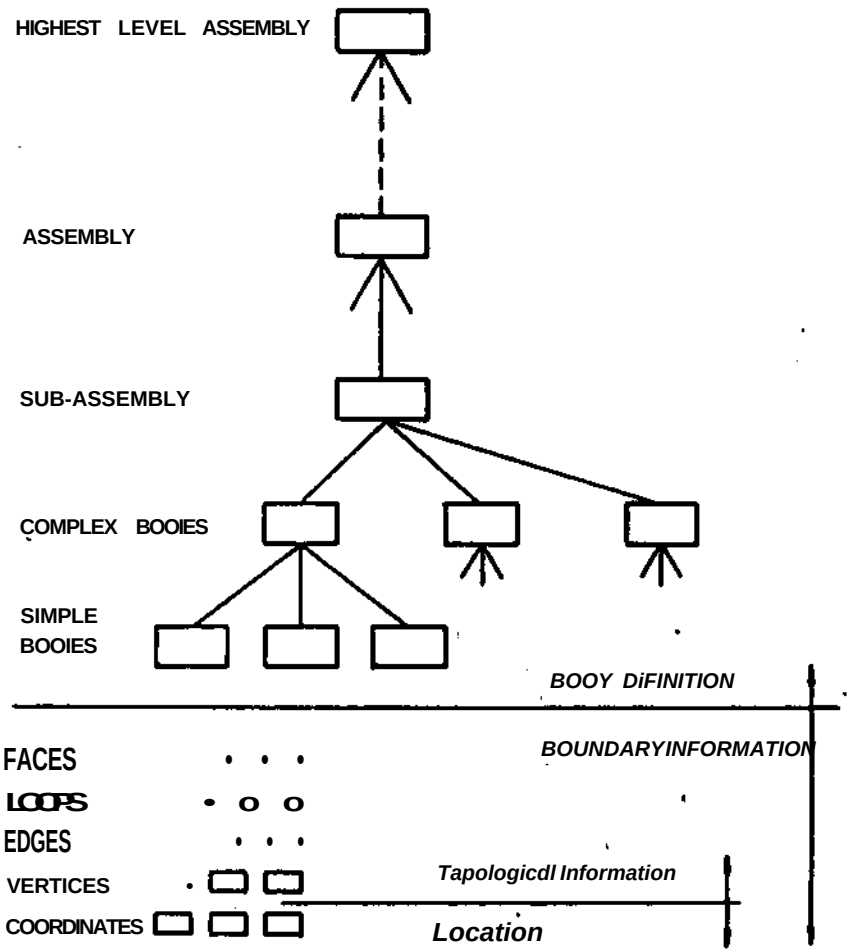


FIG. 10 THE ASSEMBLY OF FIG. 9 AS A HIERARCHY WITH CONSTRAINT RELATIONS AT EACH LEVEL. **Constraint relations are shown in broken lines.**



PIG. 11 THE 3 GENERAL LEVELS OF INFORMATION.