Naval Research Laboratory
Stennis Space Center, MS 39529-5004

VPF+: A Vector Product Format Extension Suitable for Three-Dimensional Modeling and Simulation

Sponsored by the Defense Modeling & Simulation Office

MAHDI ABDELGUERFI
ROY LADNER

University of New Orleans
New Orleans, LA

KEVIN B. SHAW
MIYI J. CHUNG
RUTH WILSON

Mapping, Charting, and Geodesy Branch
Marine Geosciences Division

August 14, 1998

19981026 061

REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE August 14, 1998	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

VPF+: A Vector Product Format Extension Suitable for Three-Dimensional Modeling and Simulation

5. FUNDING NUMBERS

Job Order No. 574-5908-08

Program Element No. 0603832D

Project No.

Task No.

Accession No.

6. AUTHOR(S)

Mahdi Adbelguerfi*, Roy Ladner*, Kevin B. Shaw, Miyi J. Chung, and Ruth Wilson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Marine Geosciences Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/7441--98-9683

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Imagery and Mapping Agency
Mail Stop P-23
12310 Sunrise Valley Drive
Reston, VA 20191-3449

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*University of New Orleans, New Orleans, LA

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

With support from the Defense Modeling and Simulation Office and the National Imagery and Mapping Agency's (NIMA) Terrain Modeling Program Office, the Digital Mapping, Charting, and Geodesy Analysis Program (DMAP) has investigated an extension to NIMA's current Vector Product Format (VPF) that would benefit the Modeling and Simulation community. In its current form, VPF's winged-edge topology is documented as not being capable of modeling a wide range of three-dimensional (3D) objects that may be transmitted and received through the Synthetic Environment Data Representation and Interchange Specification (SEDRIS). This range of objects includes non-manifold objects found in integrated, 3D synthetic environments. DMAP therefore proposes VPF+, an extension to VPF that provides for georelational modeling in 3D and that is SEDRIS capable. VPF+ adds a new level of topology called Level 4 Full 3D Topology (Level 4). The topologic information encompasses the adjacencies involved in 3D manifold and non-manifold objects, and is described using a new, extended Winged-Edge data structure. This data structure is referred to as *"Non-Manifold 3D Winged-Edge Topology."* Level 4 also adds a new 3D_Object feature class that is intended to capture a wide range of 3D objects. These features are further defined to be either Well Formed or Not Well Formed, with Well Formed 3D_Object features having additional optional topologic information to improve software performance. Finally, Level 4 implements no changes that alter VPF's Level 0 through Level 3 topology.

14. SUBJECT TERMS

modeling and simulation, 3D, data base, object oriented, topology

15. NUMBER OF PAGES 60

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Same as report

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

CONTENTS

EXECUTIVE SUMMARY

With support from the Defense Modeling and Simulation Office and the National Imagery and Mapping Agency's (NIMA) Terrain Modeling Program Office, the Digital Mapping, Charting, and Geodesy Analysis Program (DMAP) has investigated an extension to NIMA's current Vector Product Format (VPF) that would benefit the Modeling and Simulation community. In its current form, VPF's winged-edge topology is documented as not being capable of modeling a wide range of three-dimensional (3D) objects that may be transmitted and received through the Synthetic Environment Data Representation and Interchange Specification (SEDRIS). This range of objects includes non-manifold objects found in integrated, 3D synthetic environments. DMAP therefore proposes VPF+, an extension to VPF that provides for georelational modeling in 3D and that is SEDRIS capable. VPF+ adds a new level of topology called Level 4 Full 3D Topology (Level 4). The topologic information encompasses the adjacencies involved in 3D manifold and non-manifold objects, and is described using a new, extended Winged-Edge data structure. This data structure is referred to as *"Non-Manifold 3D Winged-Edge Topology."* Level 4 also adds a new 3D_Object feature class that is intended to capture a wide range of 3D objects. These features are further defined to be either Well Formed or Not Well Formed, with Well Formed 3D_Object features having additional optional topologic information to improve software performance. Finally, Level 4 implements no changes that alter VPF's Level 0 through Level 3 topology.

VPF+: A VECTOR PRODUCT FORMAT EXTENSION SUITABLE FOR THREE-DIMENSIONAL MODELING AND SIMULATION

1.0 INTRODUCTION

This report describes the data structures and data organization of *VPF+* (Vector Product Format+). VPF+ is an extension to traditional VPF [1] that provides three-dimensional (3D) georelational modeling and that is SEDRIS (Synthetic Environment Data Representation and Interchange Specification) capable. That is, VPF+ is designed with the capability of modeling a wide range of objects that can be encountered in a SEDRIS transmittal (App. A). VPF+ is not, however, limited by SEDRIS. VPF+ does, for example, allow for the definition of specific various relationships in explicit 3D topology that SEDRIS does not.

Modeling in 3D in VPF+ is provided by a new level of topology called *Level 4 Full 3D Topology*. Level 4 full 3D topology (or simply Level 4) uses a boundary representation (B-rep) method. B-rep models 3D objects by describing them in terms of their bounding entities and by topologically orienting them in a manner that enables the distinction between the object's interior and exterior.

Consistent with B-rep, the representational scheme of Level 4 includes both topologic and geometric information. The topologic information encompasses the adjacencies involved in 3D manifold and non-manifold objects, and is described using a new, extended winged-edge data structure. This data structure is referred to as *"Non-Manifold 3D Winged-Edge Topology."* The geometric information includes node coordinates and face and edge orientation. Although we restrict ourselves to planar geometry, curved surfaces can also be modeled through the inclusion of parametric equations for faces and edges as associated attribute information.

The remainder of this report is organized as follows. Section 2.0 describes non-manifold objects and explains why a new data structure is needed to represent these objects. Section 3.0 explains non-manifold 3D winged-edge topology. Section 4.0 pictorially represents this new data structure in both a relational diagram and object model. Section 5.0 provides an example of the workings of the data structure through an illustration with related data tables. Section 6.0 covers the topological sufficiency of the data structure. Section 7.0 discusses the feature level including, in particular, the introduction of *3D_Object Features* and the definition of these as being either *Well Formed* or *Not Well Formed*. It is shown that this distinction allows for the storage of optional topologic information to improve software performance. Cross-tile topology is discussed in Sec. 8.0 and Sec. 9.0 provides a 3D spatial indexing scheme. The results and future work are discussed in Sec. 10.0.

The research on which this report is based is provided in the appendices. Appendix A summarizes the features of three data structures capable of resolving some of the ambiguities found in the representation of non-manifold objects. Appendix B provides an overview of SEDRIS by (1) introducing the SEDRIS project and explaining those portions of the SEDRIS object model relevant

to this research and (2) discussing significant differences between VPF and SEDRIS. Appendix B has been provided not only for completeness of the research conducted, but also as an aid to those of the VPF community who may not be familiar with SEDRIS. It is anticipated that many who review this research will be familiar with the VPF standard [1]. For that reason, a summary of the salient features of VPF has not been provided. Appendix C offers a summary of the modified Raumbaugh notation used in the SEDRIS object model, a tool that should prove helpful in understanding App. B.

2.0 NON-MANIFOLD OBJECTS

A non-manifold object is one in which the neighborhood around some of its points is not homeomorphic to a disk [4]. The surface cannot be locally deformed into a plane without tearing it [2,4]. The non-manifold objects given consideration in Level 4 are ones with non-manifold edges and ones with non-manifold nodes. Figure 1 shows three examples of non-manifold edges. Figure 1(a) represents three faces connected along a common edge (more than three are possible). Figure 1(b) shows a "dangling" edge (an edge adjacent to no faces). Figure 1(c) shows a face "dangling" from an open box (three of its edges are adjacent to no face). Finally, Fig. 1(d) shows a non-manifold node in which two objects are connected solely at a common node. The node connecting the dangling edge in Fig. 1(b) is also an example of a non-manifold node.

Non-manifold objects are necessarily included in Level 4 topology since this class of objects may be encountered in a SEDRIS transmittal (see App. B). SEDRIS would allow, for example, the representation of the objects shown in Fig. 2(a) and (b). Figure 2(a) shows a building on a terrain surface with an antenna attached to the roof. Two non-manifold conditions are present. First, each edge along the base of the building is connected to three faces. Second, the antenna is a "dangling" edge, not adjacent to any faces.

Figure 2(b) shows a building with an internal face dividing the building into two floors. Each edge bordering the second floor is adjacent to three faces – the face forming the second floor and two faces making up the bounding walls of the building.

VPF's winged-edge data structure is insufficient to maintain all adjacency relations present in the objects shown in Fig. 1. With non-manifold objects, an edge may not be adjacent to exactly two faces. Instead, it may be adjacent to more than two faces (Fig. 1(a)), no faces (the dangling edge
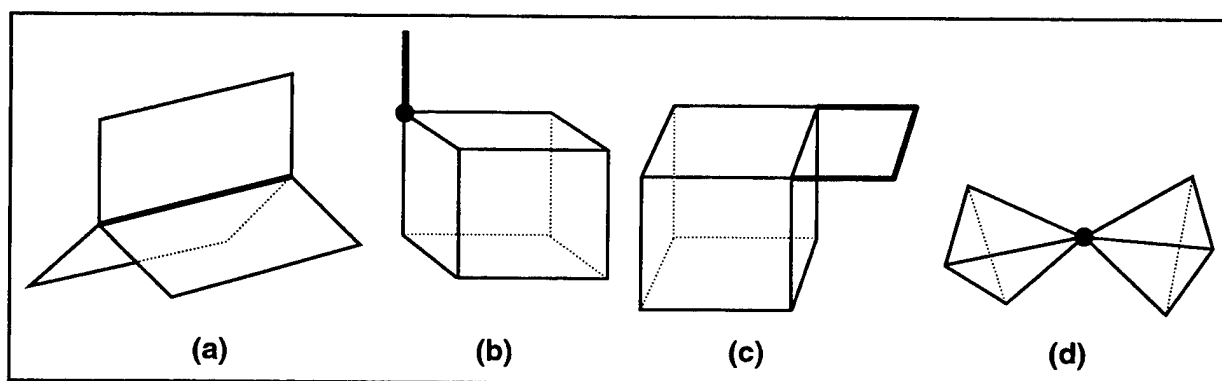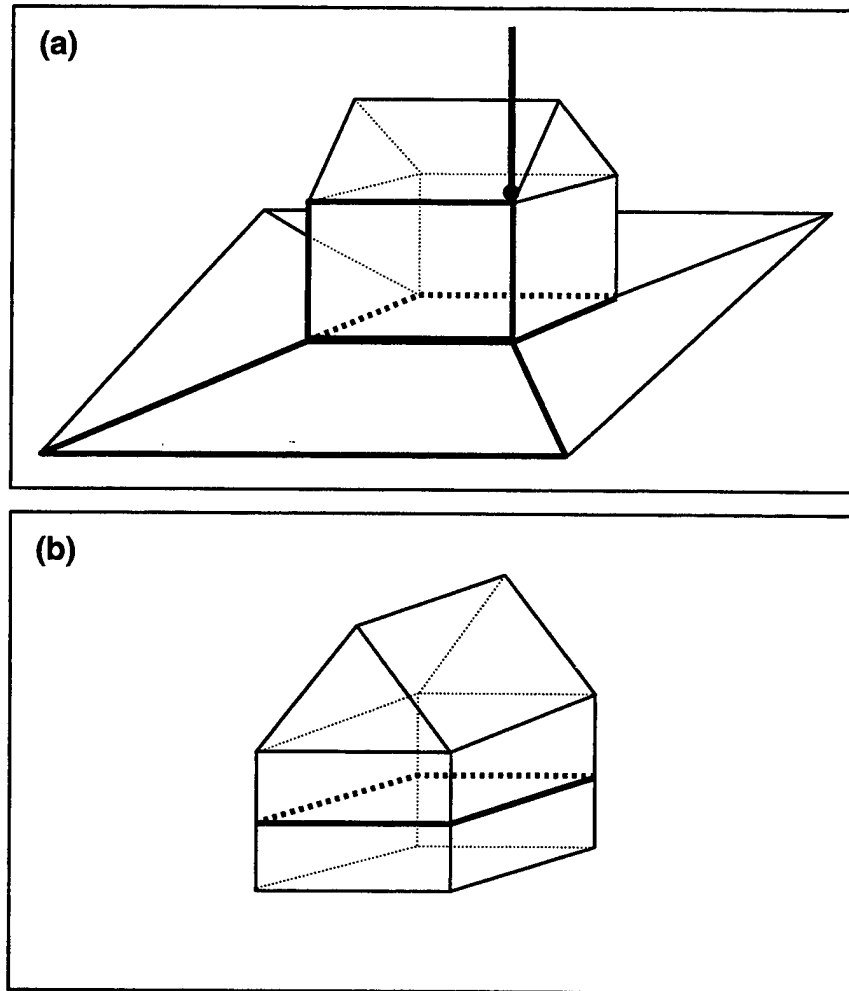

Fig. 1 — Four non-manifold objects

Fig. 2 — (a) Example of object with two non-manifold conditions and (b)
identifies a double-sided face inside a 3D object (bold-face line)

in Fig. 1(b)), or one face (one of the outer edges of the dangling face in Fig. 1(c)). Additionally, a connected node may be adjacent to edges in two or more different objects (Fig. 1(d)) or an object and a dangling edge (Fig. 1(b)).

Several alternate data structures capable of maintaining the adjacency relationships found in manifold and non-manifold objects were examined but could not be directly implemented in Level 4 (App. A). Our primary area of concern is modeling synthetic environments. These other data structures have a different application area, solid modeling, making them inconsistent with the winged-edge topology concepts found in the VPF standard [1]. However, they did provide a theoretical basis for the primitives and structures necessary to make Level 4 a logical extension of the VPF standard [1].

3.0 DATA STRUCTURE OVERVIEW

The non-manifold 3D winged-edge data structure introduces a new structure called *Efaces* to resolve the ambiguities resulting from the absence of a fixed number of faces adjacent to an edge.

Efaces describe a use of a face by an edge and allows maintenance of the adjacency relationships between an edge and 0, 1, 2, or more faces incident to an edge. An edge's left and right edges are replaced by the *Next_Edge_on_EFace*, which is always ordered to provide the next edge around the face. Additionally, VPF's connected node table is modified in Level 4 to allow for non-manifold nodes (Fig. 1(b, d)). This requires that a node point to one edge in each object connected solely through the node and to each dangling edge, and it allows the retrieval of all edges and all faces in each object and the retrieval of all dangling edges connected to the node.

Unlike VPF's Level 3 topology, the "universe face" is absent in Level 4 since Level 4 is primarily intended for 3D modeling. Additionally, since Level 4 topology is intended to model in 3D, faces may be one-sided or two-sided. A two-sided face, for example, might be used to represent a building with one side used for the outside of the building and the other side for the inside of the building. Feature attribute information would be used to render the two different surface textures and color. A one-sided face might then be used to represent the terrain surface adjacent to the building. Additionally, orientation of the interior and exterior of 3D objects is organized in relation to the normal vector of faces forming the surface boundary of closed objects.

Faces are also allowed to be embedded within a 3D object in Level 4. Figure 2(b) shows an example of a building with an embedded double-sided face dividing the building into two floors. Other double-sided faces could also be inserted to divide each floor into separate rooms.

Level 4 is a full 3D topology that is capable of representing comprehensive, integrated 3D synthetic environments. Such an environment can include objects generally associated with the terrain surface (buildings, roads, and motor vehicles for example). Additionally, it can include objects that are not attached to the terrain but are rather anchored at some significant elevation point above the terrain surface (weather systems and satellites, for example) or below a water body's surface (such as suspended acoustic objects).

3.1 Non-Manifold 3D Winged-Edge Topology

This portion of Sec. 3.0 describes the structures used by non-manifold 3D winged-edge topology in Level 4. The Level 4 topologic structures are eface, face, edge, connected node, entity node, ring, and text. Figure 3 shows the VPF+ primitive directory for Level 4 topology. Since Level 4 treats ring, entity node, and text similar to traditional VPF Levels 0 through 3, further discussion of these is omitted. The remainder, eface, face, edge, and connected node are described below.

Primitives (such as shells, typically associated with 3D topology) have been intentionally excluded from Level 4. The addition of shells as topological primitives would have been a logical extension from 2D to 3D objects since a shell is a generalization of the notion of ring as used in VPF. Whereas a ring is a mechanism that models disconnected graphs within a surface, a shell is a mechanism that models disconnected graphs between surfaces. Shells have advantages such as logically organizing adjacency relationships in 3D objects. However, these advantages are outweighed by additional complexities that arise in connection with tiling in VPF. Tiling requires that a primitive that crosses tile boundaries be split into two new primitives. Cross-tile topology is then used to retrieve the original primitive. Splitting a shell that crosses a tile boundary would create two new shells with the addition of multiple faces, edges, and nodes, making the cross-tile topological retrieval more complex and time consuming. To avoid these difficulties, a new feature is introduced, referred to as the *3D_Object Feature*. 3D_Object Features maintain adjacencies yet also model real "things." Additionally, the omission of shells is in keeping with the spirit of SEDRIS, which does not define 3D topological primitives such as shells (App. A).
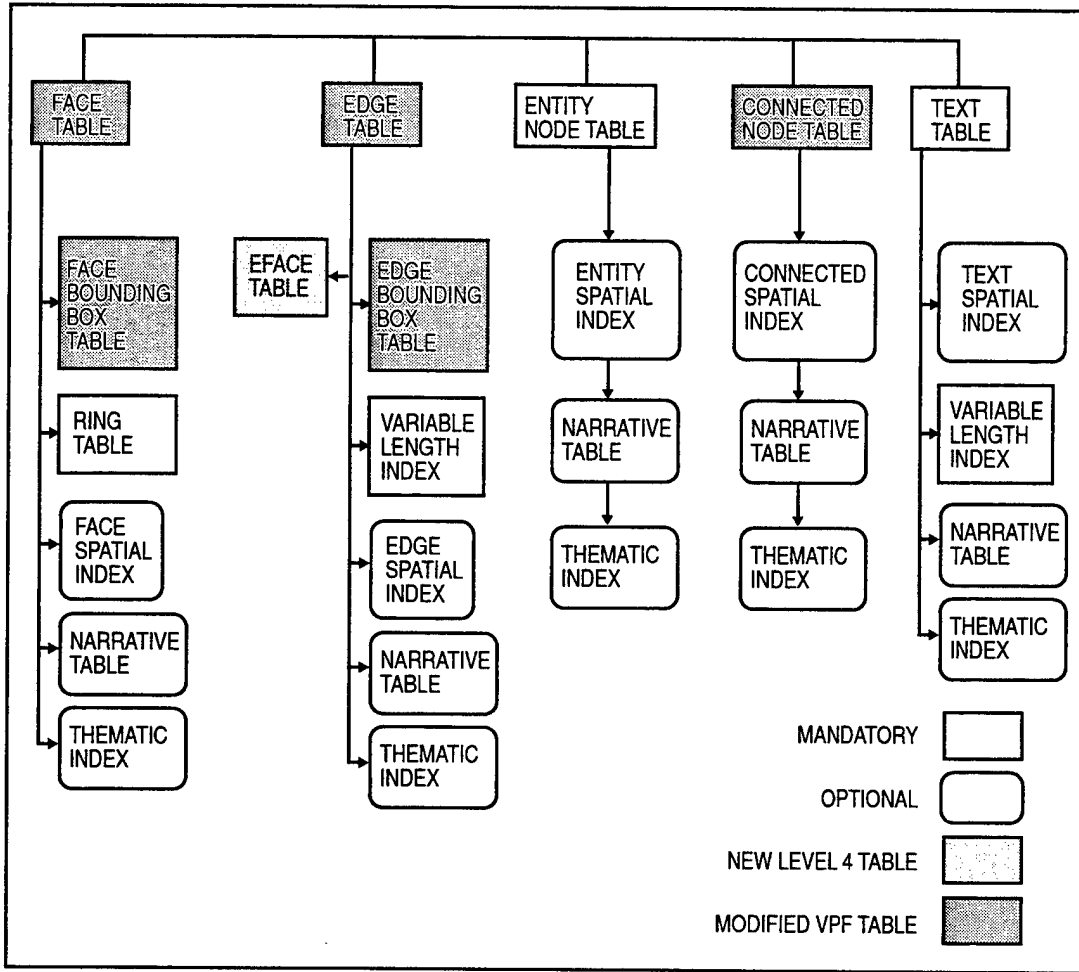
Fig. 3 — VPF+ primitive directory contents, Level 4 full 3D topology

3.2 EFace

An eface describes a use of a face by an edge (Table 1). Efaces maintain the adjacency relationship between an edge and multiple faces connected along that edge. This is accomplished by linking each edge to all faces connected along the edge through a circular linked list of efaces. Each eface in the list identifies the face it is associated with, the next eface in the list, and the Next_Edge_on_Eface. The Next_Edge_on_Eface makes it possible to find the "next" edge on a face from a given edge. Each face can also be found in more than one eface. With the observer looking in the same direction as the edge direction, efaces are radially ordered in the linked list in a clockwise direction about the edge as shown in Fig. 4. The purpose for the ordering is to make traversal from one face to the radially closest adjacent face a simple list operation.

Each eface in the eface table has an entry for face id, eface id, and edge id. Each of these serves as a foreign key into the face, eface, and edge tables, respectively. The face id indicates the face that the eface is identified with. The eface id points to the Next_Eface following the current eface in the ordered, circularly linked list of efaces adjacent to a particular edge. The edge id points to the Next_Edge_on_Eface. The Next_Edge_on_Eface is always determined by a clockwise ordering of edges about the eface and is always the next edge on the eface clockwise from the current edge.

Table 1 — EFace Table Definition

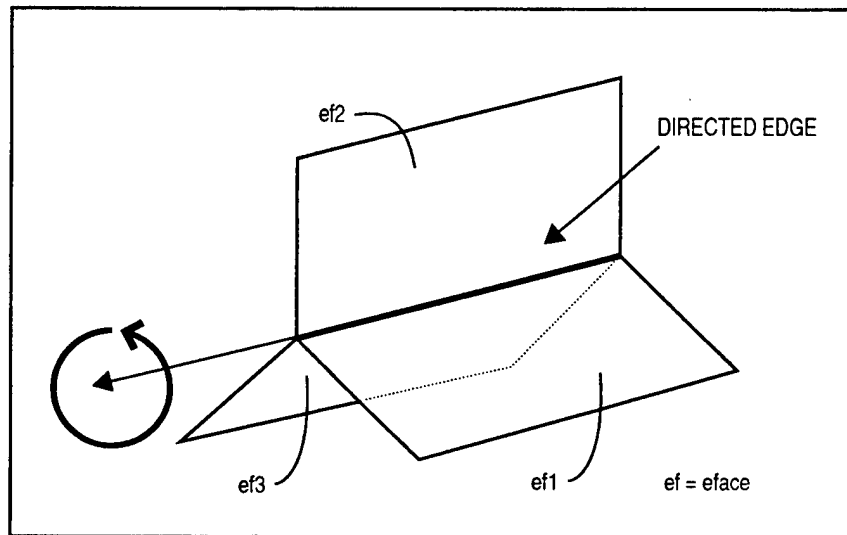| COLUMN NAME	DESCRIPTION	COLUMN TYPE	KEY TYPE	OP/MAN
ID	Row id	I	P	M4
Face	Face id	I	N	M4
Next_Eface	Eface id	I	N	M4
Next_Edge_on_Eface	Edge id	I	N	M4

Fig. 4 — Ordering of efaces about a directed edge

In both cases, the reference point is the observer standing above the face that the referenced edge is adjacent to. For all efaces generated on each edge about a given face, the reference point must remain static in reference to the face. For edge e1, Fig. 5 identifies each face/eface and shows the Next_Edge_on_Eface for each eface.

3.3 Edge

In VPF's winged-edge topology [1], each edge is related to one start node, one end node, one left face, one right face, one left edge, and one right edge. However, these relationships may not hold in a non-manifold domain.

In the non-manifold 3D winged-edge topology, edges are topologically linked to start and end nodes and to the first and last efaces in a circularly linked list of efaces. As mentioned, each eface in turn points to the face it is associated with, the next eface, and the Next_Edge_on_Eface.

The entries in the edge table (Table 2) for start and end node id are foreign keys to the connected node table referencing the edge's start and end nodes. The entries for the eface id are foreign keys to the eface table referencing the first and last efaces adjacent to the edge. Following the links in the eface table identifies each eface in the ordered, circularly linked list of efaces around the edge.
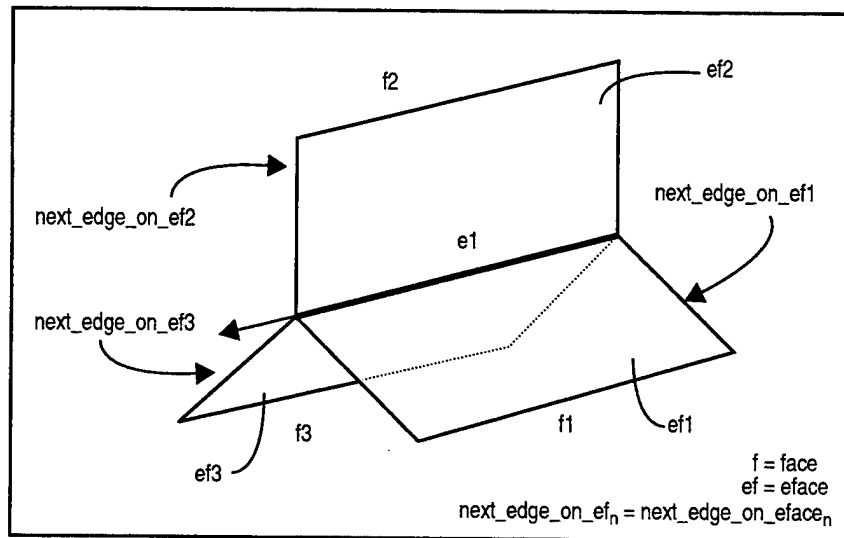
Fig. 5 — Relationship of face, eface, and Next_Edge_on_Eface for edge e1

Table 2 — Edge Table Definition

COLUMN NAME	DESCRIPTION	FIELD TYPE	KEY TYPE	OP/MAN
ID	Row id	I	P	M
**.LFT_ID	Feature id	I	N	OF
Start_Node	Start Node id	I	N	M1–4
End_Node	End Node id	I	N	M1–4
First_Eface	Eface id	I	N	M4
Last_Eface	Eface id	I	N	M4
Coordinates	Coordinates	Z/Y,*	N	M

Note: (**) indicates a place holder for the Line Feature class name; the (Z/Y,*) identifies a three-coordinate string.

Figure 6 shows a cross-section of three adjacent faces, f1, f2, and f3, adjacent to common edge e1. Edge e1 is oriented in the direction noted by the arrow on the edge. Three efaces are identified by ef1, ef2, and ef3. The relationship of edge e1 to each of the adjacent faces is shown by the arrows from e1 to the first eface, ef1, and the last eface, ef3. Eface ef1 points to face f1 and the next eface, ef2; eface ef2 points to face f2 and the next eface, ef3. Finally, eface ef3 points to face f3 and to the first eface, ef1, completing the cycle.

When the first eface equals the last eface and both are null, a dangling edge is defined. When the first eface equals the last eface and neither is null, an edge coincident with only one face is defined.
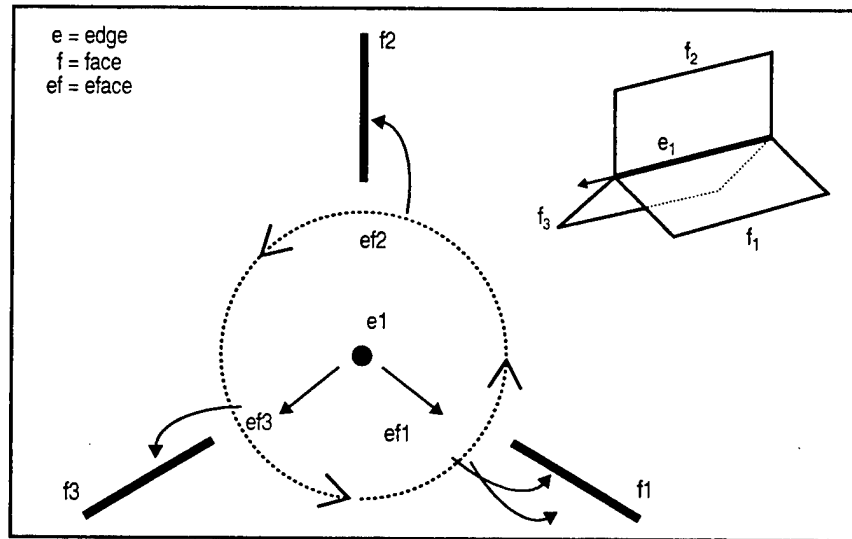
Fig. 6 — Cross-sectional view of faces sharing an edge

3.4 Face

In Level 4, faces remain planar regions as in traditional VPF [1]. The entry in the face table (Table 3) for ring id is a foreign key to the ring table identifying the ring associated with the face. The ring, in turn, identifies a starting edge, allowing the retrieval of all edges about a face when starting with only a face id.

As more fully defined in the section on features, faces can be part of either 3D_Object Features or Area Features. The entry in the **.FFT column is 1 for 3D_Object Features and 0 for Area Features. The next column gives the feature id that the face primitive is associated with.

Table 3 — Face Table Definition

COLUMN NAME	DESCRIPTION	COLUMN TYPE	KEY TYPE	OP/MAN
ID	Row id	I	P	M3–4
Ring_Ptr	Ring id	I	N	M3–4
**.FFT	Feature Type	I	N	OF4
Feature_ID	Feature id	I	N	OF4
Sides	Single/Double Sided	I	N	M4
Side_Use	Pos/Neg/Both Sides	I	N	M4
Face_Class	In/Out/Boundary***	I	N	O4
Out	Exterior of Object****	I	N	O4
Normal_Vector	Face Orientation	Z/Y,*	N	M4

Note: (**) indicates a place holder for the 3D_Object or Area Feature class name; the (***) indicates null values for faces part of Area Features and Not Well Formed 3D_Object Features; the (****) indicates null values for faces part of Area Features, non-boundary faces of Well Formed 3D_Object Features, and all faces of Not Well Formed 3D_Object Features; the (Z/Y,*) identifies a three-coordinate string.

A face may be single-sided or double-sided. Only one side of a single-sided face is used, while both sides of a double-sided face may be used. An entry of 1 in the sides column indicates double-sided and 0 indicates single-sided. The Side_Use column indicates which side of a face is used: +1 for the side facing the normal vector, −1 for the opposite side, or 0 for both sides of a double-sided face.

The Face_Class column will contain null values for faces that are part of Area Features and Not Well Formed 3D_Object Features. The Out column will contain null values for faces that are part of Area Features, non-boundary faces of Well Formed 3D_Object Features, and all faces of Not Well Formed 3D_Object Features.

Face_Class indicates whether the face is inside the 3D_Object, outside the object, or part of the surface boundary of the object. The Face_Class column may contain values of −1, 1, or 0 for inside, outside, or boundary, respectively. The Out column records the orientation of the interior and exterior of the closed portion of the object with respect to the normal vector of the boundary face. An entry of 1 indicates that the exterior of the object is in the direction of the normal vector and 0 indicates that it is in the direction opposite to the normal vector.

In addition to permitting the characterization of each side of a double-sided face, face normals have other uses. For instance, face normals are typically needed at run time for shading calculations. Therefore, the availability of a face normal vector will obviate the need for calculation at run time, thereby improving performance.

Level 4 topology imposes several restrictions on faces that are comparable to those of Level 3. While surfaces and objects are allowed to be non-manifold, individual faces of surfaces are required to be manifold. This prevents a face from self-intersecting, except along its boundary. Edges may only intersect faces at their boundary.

3.5 Connected Node

In Level 4 topology, Connected_Nodes may be topologically linked to more than one edge connected to the Connected_Node. When the Connected_Node is non-manifold (e.g., two or more objects or an object and dangling edge connected solely at the single Connected_Node), then the Connected_Node is related to one edge in each object connected at the connected node and to each dangling edge.

The connected node normal vector is included as an optional feature in Level 4. Some graphics formats such as VRML (Virtual Reality Modeling Language) allow the option of binding normals on a per-vertex basis to create shading effects that decrease the faceted nature of an object. Calculating these vertex normals as a preprocessing step can improve run time performance.

The edge id in the $edge_n$ column of the connected node table (Table 4) is a foreign key to the edge table and identifies each edge that the connected node is related to as described above. The subscript is used to indicate that there may be more than one edge related to a given connected node.

An example of a representation requiring multiple edge entries for a given connected node is given in Fig. 7. In Fig. 7, node p is related to edge e1 in object A and e7 in object B. The corresponding edge table would record both edges related to node p.

Table 4 — Connected Node Table

COLUMN NAME	DESCRIPTION	COLUMN TYPE	KEY TYPE	OP/MAN
ID	Row id	I	P	M
**.PFT_ID	Feature id	I	N	OF
Containing_Face	Null	X	N	O
Edge$_n$	Edge id of One Edge in Each Object & Dangling Edge Connected at the Node	K/I	N	M4
Normal_Vector	Node Normal Vector	I	N	O4
Coordinate	Coordinate	Z/Y,*	N	M

Note: (**) indicates a place holder for the Point Feature class name; the (Z/Y,*) identifies a three-coordinate string.
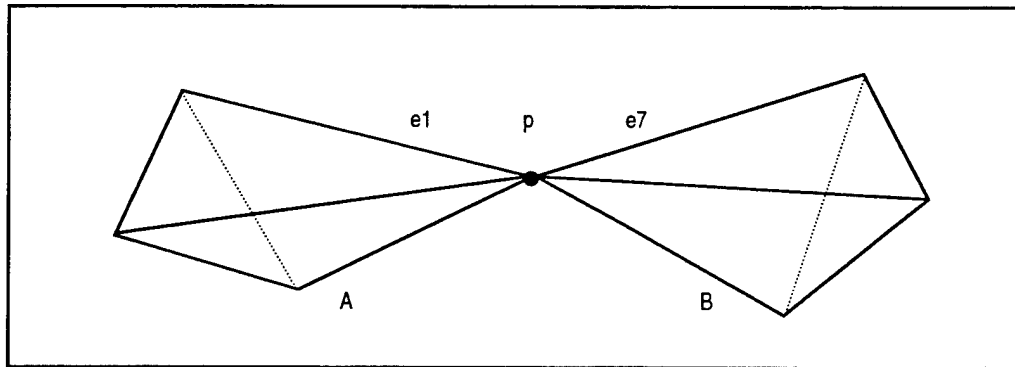

Fig. 7 — Two objects "A" and "B" connected at non-manifold node "p"

3.6 Minimum Bounding Box

Traditional VPF requires a minimum bounding rectangle record for each record in an edge or face primitive table [1]. Level 4 requires a minimum bounding box record (Table 5) for each edge or face primitive. This table adds ZMIN and ZMAX values to the minimum bounding rectangle table of traditional VPF.

Table 5 — Minimum Bounding Box Table

COLUMN NAME	DESCRIPTION	COLUMN TYPE	KEY TYPE	OP/MAN
ID	Row id	I	P	M
XMIN	Minimum x Coordinate	F/R	N	M
YMIN	Minimum y Coordinate	F/R	N	M
ZMIN	Minimum z Coordinate	F/R	N	M4
XMAX	Maximum x Coordinate	F/R	N	M
YMAX	Maximum y Coordinate	F/R	N	M
ZMAX	Maximum z Coordinate	F/R	N	M4

4.0 DATA STRUCTURE RELATIONSHIPS AND OBJECT MODEL

The data structure relationships of the non-manifold 3D winged-edge topology are summarized in Fig. 8 and an object model is provided in Fig. 9. References to geometry are omitted.

As shown in Fig. 8, each Connected_Node is related to one edge in each manifold object to which the node is attached and to each dangling edge connected to the node. Each edge is related to its start and end nodes and to its first and last efaces. Each eface is related to its face, the Next_Eface in the ordered, circularly linked list of efaces that the eface is a member of, and to the Next_Edge_on_Eface. The face, in turn, is linked to a ring that is related to its starting edge. An object model for non-manifold 3D winged-edge topology is given in Fig. 9. The SEDRIS modified Raumbaugh notation is used.

5.0 LEVEL 4 TOPOLOGY EXAMPLE

An example of the Level 4 topology is given using Fig. 10. Two non-manifold conditions are shown in Fig. 10: each edge forming the base of the building is connected to more than two faces and there is a dangling edge representing an antenna extending from the roof of the building.

Tables 6–10 illustrate some of the adjacency relationships for Fig. 10. References to geometry are omitted. Additionally, not all table entries use id's for the sake of clarity.

The sample edge table, Table 6, contains entries for the edge, Start_Node, End_Node, First_Eface, and Last_Eface id's for some of the edges. Edge e1, for example, has a reference to a First_Eface, ef1, and to a Last_Eface, ef3. Following the pointer from edge e1 to the sample eface table, Table 7, identifies the first face adjacent to edge e1 as face f3. The Next_Eface is identified as eface ef2, and the Next_Edge_on_Eface as edge e8. The identity of each face adjacent to edge e1 can be found by following the links in the eface table.
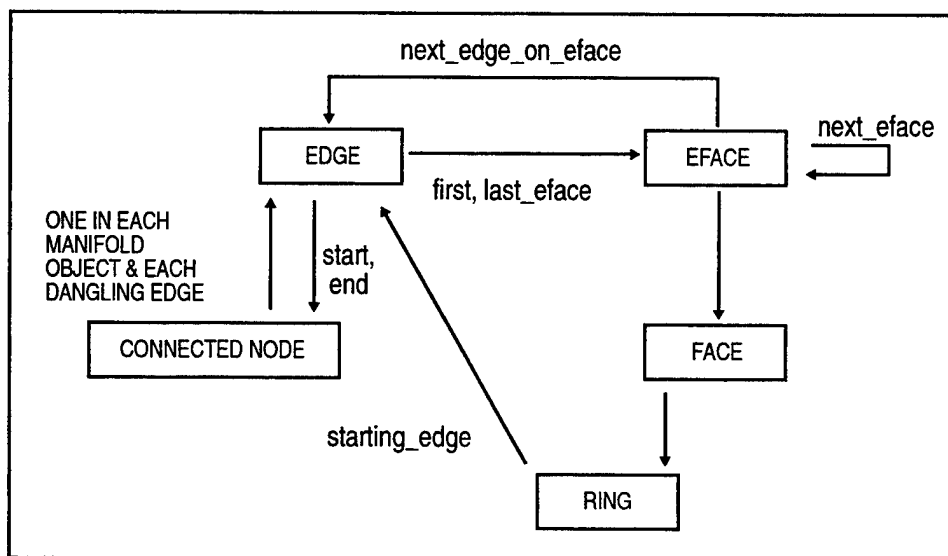

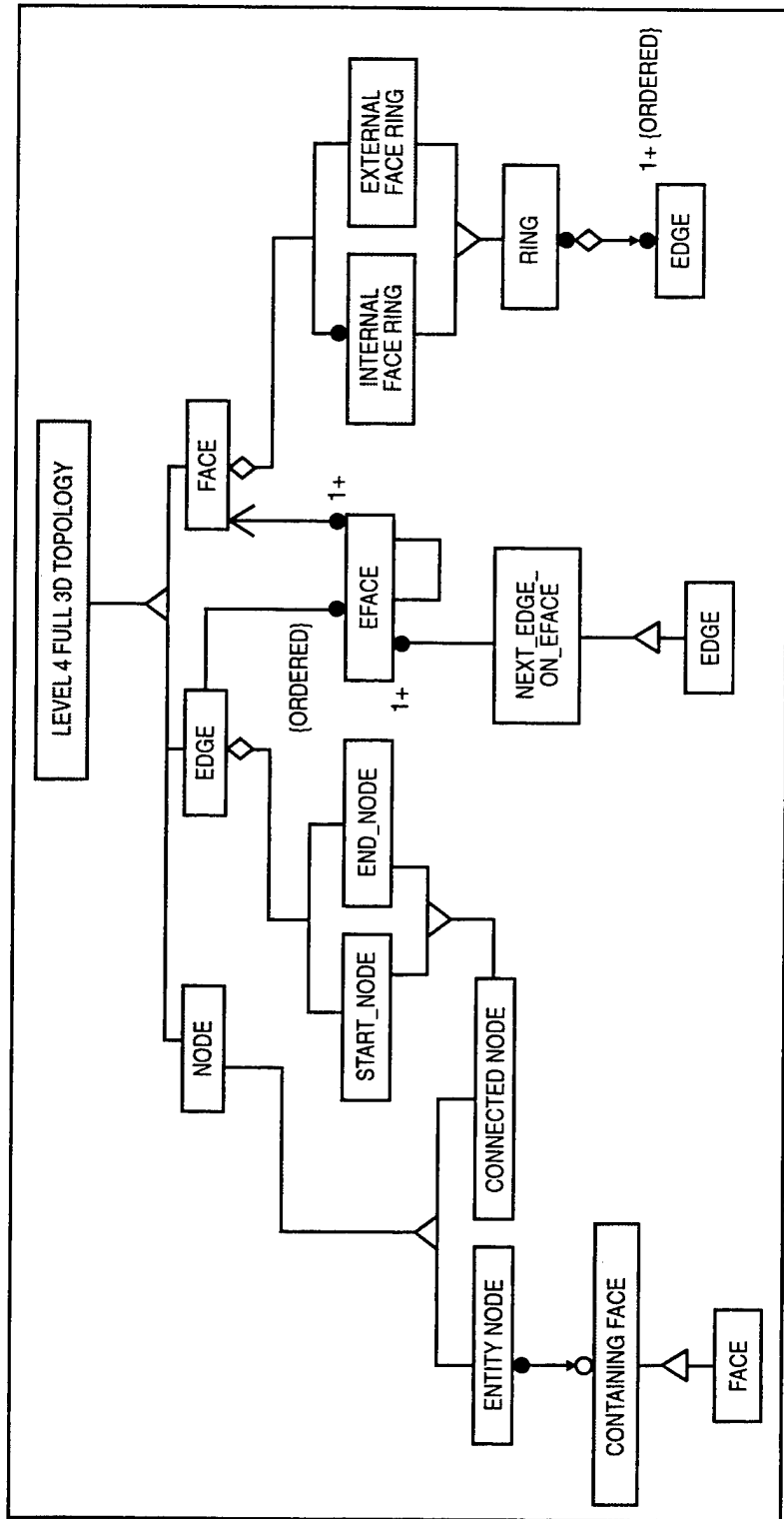Fig. 8 — Non-manifold 3D winged-edge structure relationships
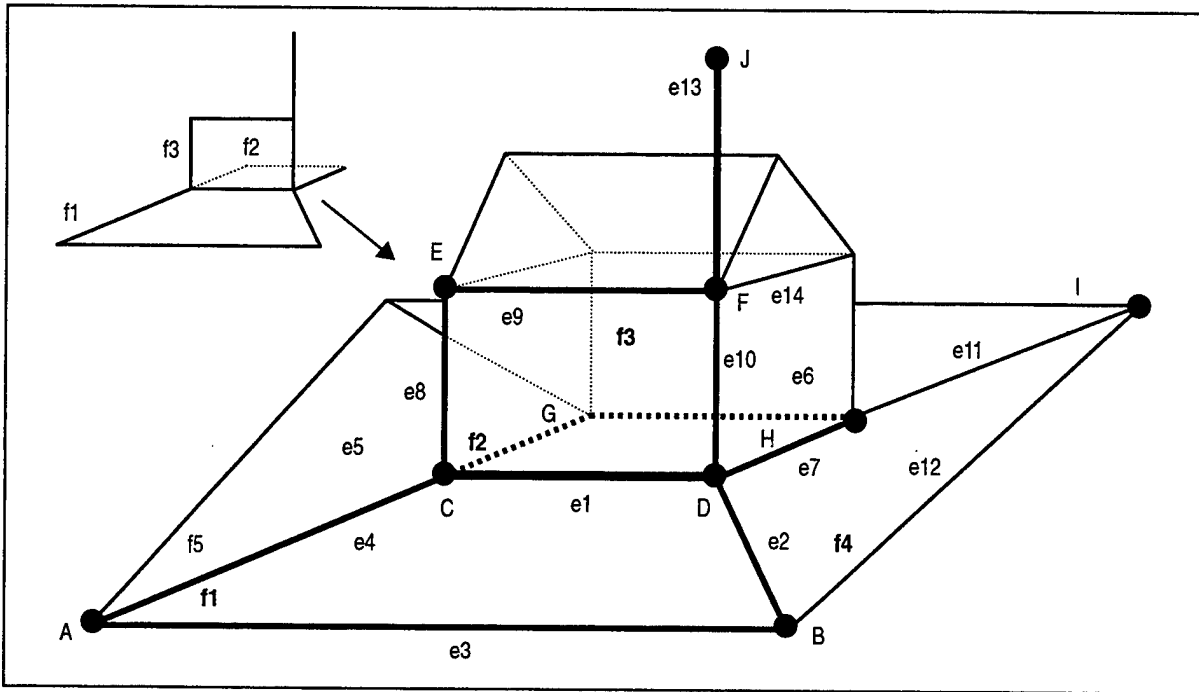
Fig. 9 — Level 4 full 3D topology object model

Fig. 10 — Non-manifold 3D winged-edge example

Table 6 — Sample Edge Table for Fig. 10

EDGE	START_NODE	END_NODE	FIRST_EFACE	LAST_EFACE
e1	C	D	ef1	ef3
e2	D	B	ef4	ef5
e3	B	A	ef6	ef6
e4	C	A	ef7	ef8
e5	C	G	omitted	omitted
...	...	...	...	...
e7	D	H	omitted	omitted
e8	C	E	ef11	ef12
e9	F	E	ef13	ef14
e10	D	F	ef9	ef10
e13	F	J	null	null
...	...	...	...	...

Table 8 shows the sample face table for some of the faces shown in Fig. 10. The face table contains columns for the Face_ID and Ring_ID. Each face points to the ring related to that face.

Table 9 contains entries for some of the rings for Fig. 10. The ring table has columns for the Ring_ID, Face_ID, and Start_Edge for the ring. Each ring points to its face and one starting edge.

Table 7 — Sample EFace Table for Fig. 10

EFACE	FACE	NEXT_EFACE	NEXT_EDGE_ON_EFACE
ef1	f3	ef2	e8
ef2	f1	ef3	e2
ef3	f2	ef1	e5
ef4	f4	ef5	e7
ef5	f1	ef4	e3
ef6	f1	ef6	e4
ef7	f1	ef8	e1
ef8	f5	ef7	omitted
ef9	f3	ef10	e1
ef10	omitted	ef9	e14
ef11	omitted	ef12	omitted
ef12	f3	ef11	e9
ef13	omitted	ef14	omitted
e14	f3	ef13	e10
...	...	...	...

Table 8 — Sample
Face Table for Fig. 10

FACE_ID	RING_ID
f1	2
f2	3
f3	1
f4	4
f5	omitted
...	...

Table 9 — Sample Ring Table for Fig. 10

RING_ID	FACE_ID	START_EDGE
1	f3	e10
2	f1	e2
3	f2	e5
4	f4	e2
...	...	...

Some of the entries for the connected node table are shown in Table 10. For example, Table 10 shows a connected edge of node D to be e1. The other edges connected to node D can be obtained by navigating through the edge and related tables. For node F, however, Table 10 shows two connected edges, e9 and e13, since edge e13 is a dangling edge.

6.0 TOPOLOGICAL SUFFICIENCY

As described in [2,3], nine adjacency relations can be defined between the node-face-edge primitives shown in Fig. 11. The directed arcs in Fig. 11 represent ordered relations between primitives. For example, the *Edge → Node* relation implies the storage of the start and end nodes with each edge.

Table 10 — Sample Connected
Node Table for Fig. 10

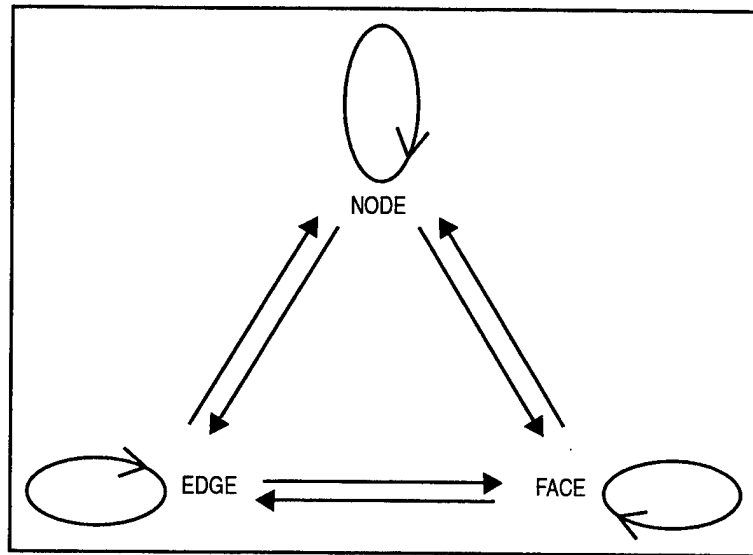| NODE_ID	EDGE$_1$	EDGE$_2$
A	e3	–
B	e2	–
C	e1	–
D	e1	–
E	e8	–
F	e9	e13
G	e6	–
H	e7	–
I	e11	–
J	e13	–
...	...	...

Fig. 11 — The nine relations between the three primitives

Table 11 lists nine access primitives that were also identified in [2,3]. These access primitives describe the retrieval of all topological adjacencies for each of the face, node, and edge primitives. A data structure that stores only a subset of all possible adjacency relations, yet can satisfy queries to all nine access primitives, is said to be topologically sufficient.

VPF's winged-edge data structure satisfies queries to all nine access primitives. The non-manifold 3D winged-edge data structure also satisfies queries to all of the access primitives shown in Table 11 as explained below. Many solutions may exist for some queries. The solutions mentioned below do not purport to be the most efficient, as all formal issues related to space considerations and time efficiencies are currently being investigated.

- AP1: First, all edges around a face are identified (AP2). Then the nodes around the face are directly retrievable from the edge table as the start and end nodes for each edge.

Table 11 — The Nine Basic Access Primitives

ACCESS PRIMITIVES (AP)	DESCRIPTION
AP1	Given face i find all n_i nodes around it
AP2	Given face i find all e_i edges around it
AP3	Given face i find all f_i faces around it
AP4	Given node i find all f_i faces around it
AP5	Given node i find all n_i nodes connected to it
AP6	Given node i find all e_i edges connected to it
AP7	Given edge i find its two extreme vertices
AP8	Given edge i find the e_i edges connected to it
AP9	Given edge i find the f_i faces intersecting at it

- AP2: The face and ring tables are traversed to identify a starting edge. The edge and eface tables are then traversed. Using face f3 in Fig. 10 as an example, the face table, Table 8, identifies ring 1. The start edge for ring 1 is edge e10 (Table 9). The first eface about edge e10 is ef9 (Table 6). Table 7 identifies ef9 as related to face f3 and the Next_Edge_on_Eface as edge e1. Following the links from edge e1 yields edges e8 and e9. All edges around face f3 are, therefore, identified as e10, e1, e8, and e9 in that order.

- AP3: First all edges around the face are identified (AP2). Then all faces intersecting at each edge are found (AP9).

- AP4: All edges connected to the node are first identified (AP6). Then all faces intersecting at each such edge are found (AP9).

- AP5: All edges connected to the node are identified (AP6). The edge table identifies each node that is opposite the given node.

- AP6: The connected node table identifies one edge in each manifold object and each edge dangling from the given node. The eface and edge tables are traversed to identify all remaining edges. Giving node D in Fig. 11 as an example, the connected node table identifies edge e1 and the algorithm described for AP8 identifies edges e1, e2, e7, e10, e4, e5, and e8. Edges e4, e5, and e8 are discarded since the edge table establishes that they are not connected to node D. This leaves edges e1, e2, e7, and e10.

- AP7: The two extreme vertices of an edge are directly retrievable from the edge table listing of the start and end nodes.

- AP8: All edges connected to an edge are found by traversing the edges of each face intersecting the search edge and also of each face intersecting any additional edges identified in the connected node table. Each edge with a start or end node identical to the original search edge is selected. When the connected node table identifies more than one edge connected to the original search edge's start or end nodes, the additional edges not previously identified would also be included as connected edges. Using edge e1 in Fig. 10 as an example, the faces intersecting at it are f1, f2, and f3 (AP9). The edges bordering face f3 are identified as e10, e1, e8, and e9. Edge e9 is discarded since it does not have a common start or end node with edge e1. Repeating the process for faces f1 and f2 yields edges e2, e7, e10, e4, e5, and e8 as all of the edges connected to edge e1.

- AP9: Finding all faces intersecting at an edge is obtained from the eface table by following the links in the linked list of efaces. Using edge e1 in Fig. 10 as an example, the edge table together with the eface table identifies faces f1, f2, and f3 as adjacent faces.

7.0 FEATURES

Traditional VPF defines five categories of cartographic features: point, line, area, complex, and text. Point, line, and area features are classified as simple features, composed of only one type of primitive. Each simple feature is of differing dimensionality: zero, one, and two for point, line, and area features, respectively. Unlike simple features, complex features can be of mixed dimensionality and are obtained by combining features of similar or differing dimension.

For Level 4 topology, VPF+ adds a new simple feature class of dimension three. The newly introduced feature, referred to as *3D_Object Feature*, is composed solely of face primitives. This new feature class is aimed at capturing a wide range of 3D objects. The eface table is also added to the structural scheme. While the ring table provides a relationship between a face and all of the

edges that the face's rings are composed of, the eface table provides a relationship between an edge and all of the faces that meet at that edge. VPF+ feature class structural schema is shown in Fig. 12.

Although 3D_Objects are restricted to primitives of one dimension, 3D_Objects of mixed dimensionality can be modeled through complex features using simple features of similar or mixed dimensionality as building blocks. The building with the antenna extending from the roof shown in Fig. 2(a) is an example of such a complex feature in Level 4. The building is a simple 3D_Object Feature and the antenna is a simple line feature.

Face primitives in Level 4 are the building blocks of Area and of 3D_Object Features. Face primitives comprising both types of features can be double- or single-sided, with Area Features generally using only one side of a double-sided face. A face in Level 4 may be part of either a 3D_Object Feature, an Area Feature, or both.

7.1 3D_Object Features

3D_Object Features are topologically 3D and are intended to model real 3D objects such as a building or a motor vehicle. Since a 3D_Object Feature is a simple feature, objects being modeled at this level are restricted to be composed of faces connected along incident edges or at non-manifold connected nodes. 3D_Object Features are allowed to meet along adjacent faces, edges, and non-manifold connected nodes. Additionally, containment among 3D_Object features is allowed.
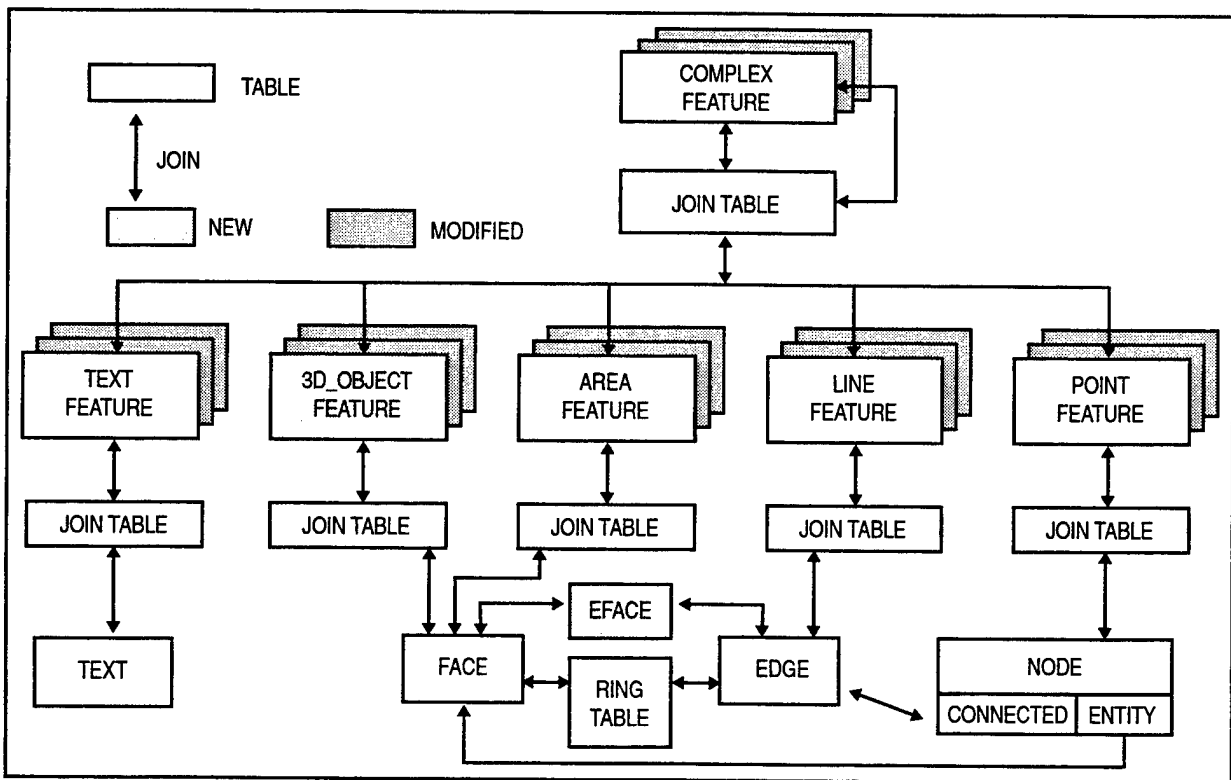

Fig. 12 — Level 4 feature class structural schema

Software performance can be improved by identifying characteristics of real 3D objects that will allow storage of optional, unambiguous topological information that may otherwise require considerable processing time to derive. Clearly, portions of numerous 3D objects form closed volumes that divide 3D space into interior, exterior, and surface regions. Optional topological information in these cases includes the classification of faces as either inside of, outside of, or part of the boundary of the 3D_Object and the orientation of the interior and exterior of the object.

3D_Object Features are, therefore, defined to be either *Well Formed* or *Not Well Formed*. A Well Formed 3D_Object Feature is structured such that:

• The 3D_Object may be manifold or non-manifold.

• One or more subsets of its composing faces forms a closed 3D volume that clearly divides 3D space into interior, exterior, and surface regions.

• This closed volume may contain internal faces.

• This closed volume may have external dangling faces.

All 3D_Object Features that do not meet these criteria are defined as Not Well Formed.

7.2 Face Classification and Object Orientation

Based on this description of Well Formed 3D_Object Features, faces may be classified unambiguously as either *Boundary*, *Inside*, or *Outside*. The subset of faces of the object forming the surface of a closed 3D volume are the boundary faces. Faces embedded within the closed portions of the 3D volumes are classified as inside. External dangling faces are classified as outside. Additionally, the interior and exterior of the closed portions of Well Formed features may be unambiguously oriented. Orientation is provided with respect to the normal vector of its boundary faces. No distinction is made in VPF+ between two or more closed 3D volumes forming part of a Well Formed 3D_Object Feature. This is left to the product specification.

Figure 13 shows an example of all three kinds of faces, as well as the interior and exterior of the 3D_Object. The inside face is delineated by the highlighted lines.

7.3 Join Table Definition

Face classification as either boundary, inside, or outside, and the orientation of the interior and exterior of the closed portion of the 3D_Object, may be found in the appropriate columns of the
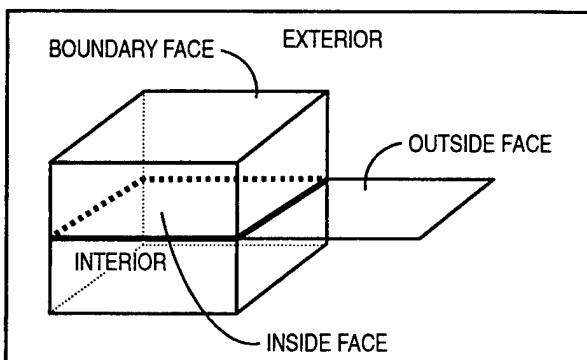

Fig. 13 — A Well Formed 3D_Object Feature

face table. In the case of one face primitive used by more than one feature or type of feature, this information may be found in the join table. The join table definition, used for both 3D_Object Features and Area Features, is shown in Table 12. An entry in the Face_Class column may be either –1, 1, or 0, indicating inside, outside, or boundary, respectively. An entry in the Out column of 1 indicates that the exterior of the object is in the direction of the normal vector of the boundary face, and 0 indicates that it is in the direction opposite of the normal vector of the boundary face. Entries in the Face_Class column are null for faces that are part of Area Features and Not Well Formed 3D_Object Features. In the Out column, entries are null for faces that are part of Area Features, non-boundary faces of Well Formed 3D_Object Features, and all faces of Not Well Formed 3D_Object Features.

7.4 Examples

Figure 14 shows an example of the use of the Well Formed 3D_Object Feature classification. In Fig. 14, two features share a common face indicated by the highlighted lines. The direction of the normal vector of the common face is given by the arrow. Since the face is found in both features, it will be listed twice in the join table. One listing will record the outside of feature "A" as being in the opposite direction from the normal vector of the shared face. The other will record the outside of feature "B" as being in the same direction as the normal vector of the shared face.

Table 12 — Face–3D_Object/Area Feature Join Table Definition

COLUMN NAME	DESCRIPTION	COLUMN TYPE	KEY	OP/MAN
ID	Row id	I	P	M
**_ID	Table 1 id	I	N	M4
Tile_ID	Tile id	S	N	MT/O
Face_ID	Table 2 id	I/S/K	N	M
Face_Class	In/Out/Boundary*	I	N	O4
Out	Exterior/Interior***	I	N	O4

Note: (**) indicates a place holder for either 3D_Object Feature or Area Feature; (*) indicates null values for faces part of Area Features and Not Well Formed 3D_Object Features; (***) indicates null values for faces part of Area Features, non-boundary faces of Well Formed 3D_Object Features, and all faces of Not Well Formed 3D_Object Features.

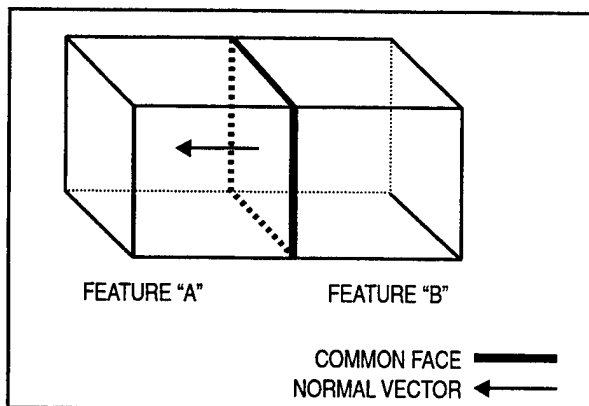FEATURE "A" FEATURE "B"

COMMON FACE ▬▬
NORMAL VECTOR ◄▬▬

Fig. 14 — Two features sharing a common face

The simplest Well Formed 3D_Object Features are closed, orientable 3-manifolds. A feature of this type divides 3D space into three regions: interior, exterior, and surface. The feature's surface region is restricted to be finite, but the interior and exterior regions are allowed to be infinite. The object is also oriented with respect to its interior and exterior regions. If the exterior region of the feature is infinite and the interior region finite, then the feature is a *Regular* one. Otherwise, the exterior is finite and the interior infinite, and the feature represents a *Void* within a 3D object. In Fig. 15, for example, the exterior feature is classified as regular. The internal one is classified as a void.

The object shown in Fig. 16 is Not Well Formed. The distinction between interior and exterior is not clearly defined, since a closed 3D volume is not formed. The object shown in Fig. 17 is Well Formed. A subset of its faces forms a closed 3D volume.

Figure 18 shows an example of a 3D_Object Feature that is Not Well Formed. Portions of three faces joining at a single edge are illustrated. All edges except the incident connected edge are adjacent to only one face. No subset of the object's faces forms a closed 3D volume dividing 3D space into interior, surface, and exterior regions.

REGULAR OBJECT

INTERNAL VOID WITHIN THE OBJECT

INDICATES DIRECTION OF EXTERIOR ⟶
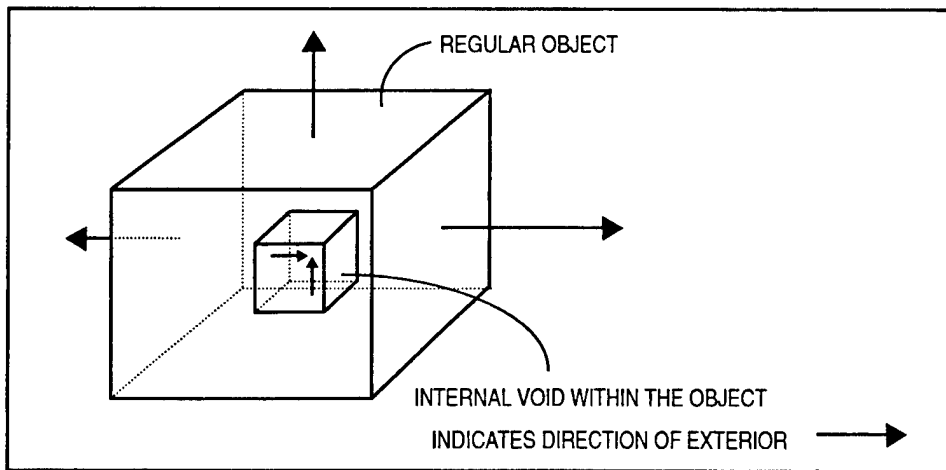
Fig. 15 — A regular object with internal void

OPEN
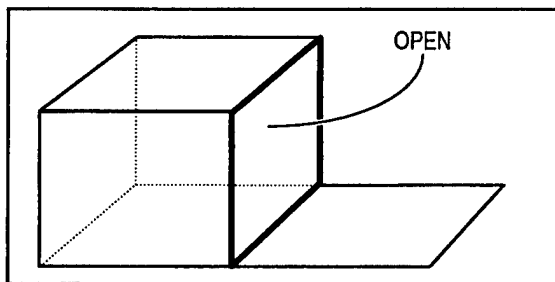
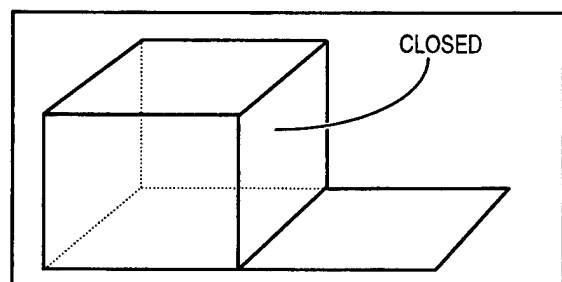Fig. 16 — A Not Well Formed 3D_Object Feature

CLOSED

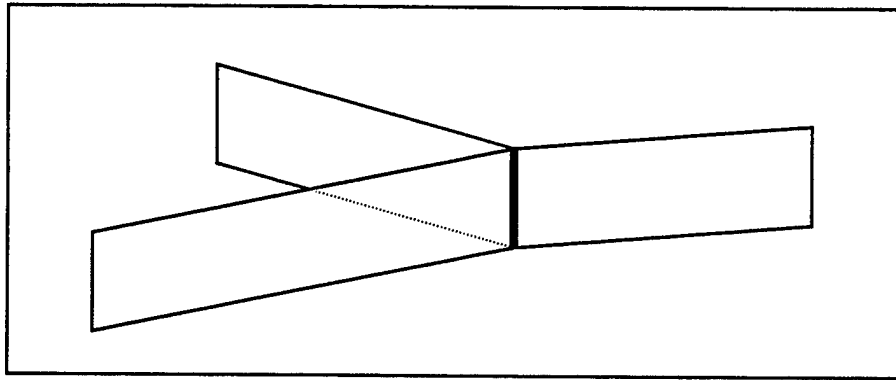Fig. 17 — A Well Formed 3D_Object Feature
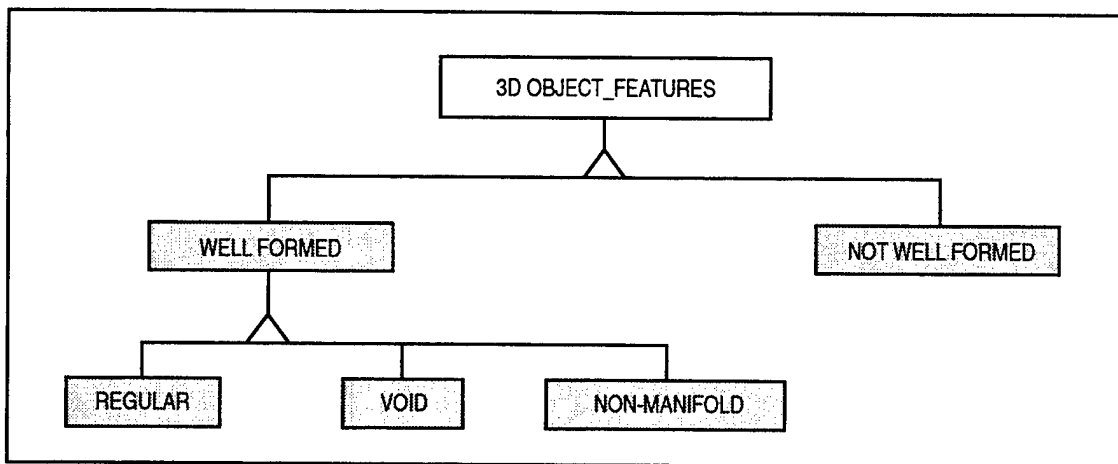
Fig. 18 — A Not Well Formed 3D_Object Feature

Fig. 19 — 3D_Object Feature classifications

7.5 3D_Object Feature Classification

The classification of 3D_Object Features previously described is summarized in Fig. 19. Shaded boxes are used to indicate abstract classifications.

7.6 Area Features

Though Area Features may geometrically exist in 3D space, they are topologically 2D and are intended to model surface area. As with 3D_Object Features, Area Features are simple features and objects being modeled at this level are restricted to be composed only of faces connected along incident edges or at non-manifold connected nodes. The faces, as mentioned, may be single-sided or double-sided, but an Area Feature will generally make use of only a single side of a double-sided face. The join table definition is shown in Table 12. Note that the entries in the Face_Class and Out columns are null for faces that are associated with Area Features.

8.0 CROSS-TILE TOPOLOGY

Tiling is the method used in VPF [1] to break up large geographic data into spatial units small enough to fit the limitations of a particular hardware platform and media. Primitives that cross-tile boundaries are split in VPF. Cross-tile topology is maintained by replacing the primitive's integer key with a triplet id. The triplet id consists of an internal reference to the primitive in the current tile and external references to the neighboring tile and a primitive within that tile. The two external references allow network navigation across tile boundaries using VPF's winged-edge topology. Level 4 employs a similar cross-tile scheme. Additionally, in Level 4 as in Level 3, the tiling scheme and the handling of features that lie on tile boundaries and text primitives that cross-tile boundaries are left to the product specification.

8.1 Cross-Tile Constructs, Traditional VPF

When tile boundaries are created, four occurrences requiring primitives to be split are possible in traditional VPF [1]:

• An edge intersects the tile boundary.

• The edge is coincident with the tile boundary.

• A face is broken by a tile boundary.

• Connected nodes occur on the tile boundary.

When primitives are split, triplet id's are created for left/right faces and left/right edges in the edge table and for first edge in the connected node table when there is a cross-tile primitive in the neighboring tile.

8.2 Cross-Tile Constructs, Level 4

Tile boundaries in Level 4 consist of planar divisions. The cross-tile constructs of traditional VPF are extended in Level 4 in accordance with the organizational scheme of non-manifold 3D winged-edge topology. Five occurrences requiring primitives to be split are possible when tile boundaries are created in Level 4 topology:

• An edge intersects the tile boundary.

• The edge is coincident with the tile boundary.

• A face is broken by a tile boundary.

• A face is coincident with the tile boundary.

• Connected nodes occur on the tile boundary.

The only new situation encountered in Level 4 is that of a face coincident with the tile boundary. The remaining four occurrences, though also encountered in Level 3, are treated differently in Level 4. When Level 4 primitives are split, a triplet id is defined for the Next_Eface and Next_Edge_on_Eface columns in the eface table and the edge column of the connected node table when there is a corresponding cross-tile primitive in the neighboring tile. The rules involved are discussed below.

• *An edge intersects the tile boundary* – An edge is always broken when it intersects a tile boundary by placing a connected node at the intersection in both tiles. The efaces related to the original

edge are also broken between the two tiles, and the efaces within each tile are circularly ordered in separate tables. All edges terminated by this connected node will have cross-tile topology if an edge exists in the neighboring tile. The cross-tile topology consists of triplet id's in the Next_Edge_on_Eface column of the eface table. The triplet id identifies the edge id of the Next_Edge_on_Eface of the present eface in the current tile, the neighboring tile id, and the id of the cross-tile edge in the neighboring tile. The cross-tile edge will be the first edge in the adjacent tile, counterclockwise from the referencing edge at the node.

Figure 20(a) shows an object consisting of three faces connected along a common edge that is divided by a tile boundary. The intersection of the faces and tile boundary is represented by the dashed lines in Fig. 20(a). Two new objects are formed as a result, one in each tile shown in Fig. 20(b) and (c). Nodes and edges have been inserted where the tile boundary divides the edges and faces of the object. These are represented by the highlighted lines and vertices. Edges e1 and e2, for example, are the result of dividing one former edge that intersected the tile boundary. For edge e1, B1 is the start node, and edge e3 is the Next_Edge_on_Eface for the eface related to face x. The cross-tile edge is edge e4 in tile 2. Table 13 gives a sample eface table entry for edge e1 in Fig. 20.

* *The edge is coincident with the tile boundary* – An edge coincident with a tile boundary occurs in both tiles. Within a given tile, the edge is related to a list of efaces that identify faces occurring within the tile. The eface in the list prior to the tile boundary has cross-tile topology if a face adjacent to the referencing edge exists in the neighboring tile. The cross-tile topology consists of
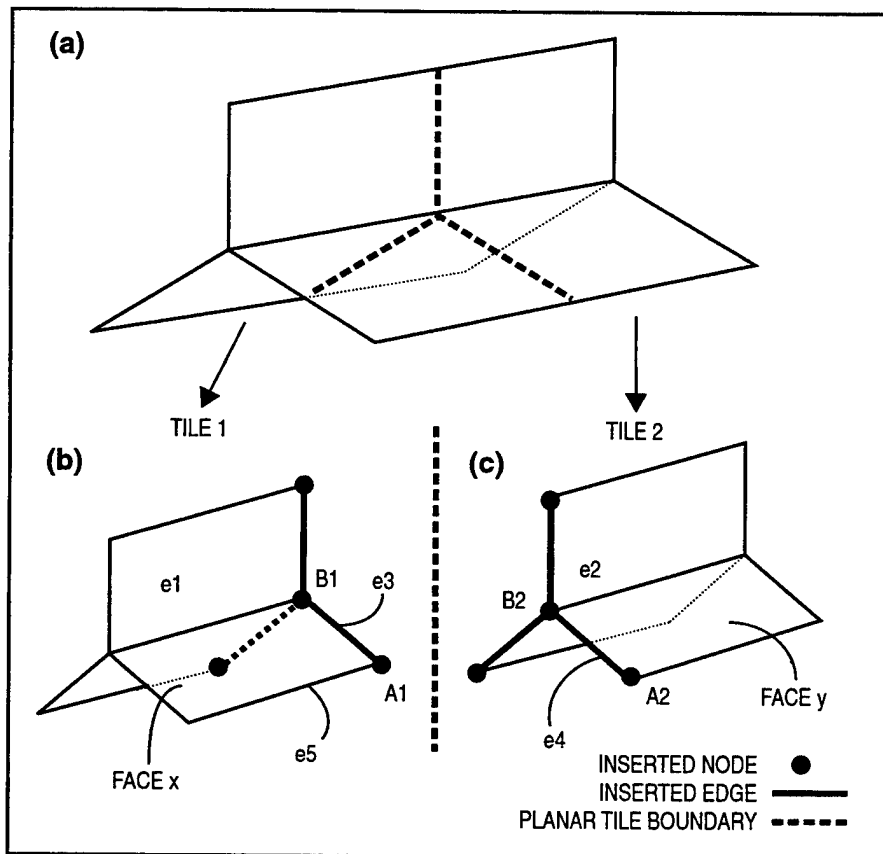

Fig. 20 — Object divided by planar tile boundary

triplet id's in the Next_Eface and Next_Edge_on_Eface columns. The triplet id in the Next_Eface column identifies the eface in the list prior to the tile boundary in the current tile, the neighboring tile id, and the next ordered eface in the linked list of efaces ordered about the coincident edge in the neighboring tile. The Next_Edge_on_Eface column contains a triplet id identical to that for an edge intersecting a tile boundary.

In Fig. 20, edges e3 and e4 are added by the division caused by the tile boundary and are treated as coincident edges. Node A1 is the start node for edge e3 and node A2 is the start node for edge e4. The eface list for edge e3 will list only one eface identifying face x. The triplet id in the Next_Eface column of the eface table will identify tile 2 and the next ordered eface (identifying face y) about the corresponding edge, e4. The Next_Edge_on_Eface column will reference edge e5 as the next edge in the current tile, tile 2, and edge e2 as the cross-tile edge.

Table 14 gives a sample eface table entry for edge e3 in Fig. 20. Similar triplet id's would be entered for all efaces adjacent to edges coincident with the tile boundary. These are the highlighted edges in Fig. 20.

• *A face is broken by a tile boundary* – When a face is broken by a tile boundary, the face is closed along the tile boundary creating multiple faces and efaces. The new edges used to close the face are treated as mentioned above.

In Fig. 20, faces x and y, for example, resulted from the division of one face by the tile boundary. Separate efaces relate to face x and face y. The edges used to close the faces are treated as coincident edges.

• *A face is coincident with the tile boundary* – In Level 4, a face may be coincident with a tile boundary. Face is coincident with a tile boundary when all edges bounding the face are on the tile boundary. Each edge forming the boundary of the face is treated as a coincident edge.

• *Connected nodes occur on the tile boundary* – This is handled similarly to traditional VPF. Connected nodes that occur on tile boundaries exist in both tiles. Each node has a triplet id for internal and external edge if an edge exists in the neighboring tile. No special treatment is needed for non-manifold nodes (see Fig. 21).

Table 13 — Sample EFace Table Entry for Edge e1 in Fig. 20

EFACE ID	FACE*	NEXT_EFACE*	NEXT_EDGE_ON_EFACE*
1	x	2, –, –	e3, tile 2, e4

*Note: Names rather than id's may be used for clarity.

Table 14 — Sample EFace Table Entry for Edge e3 in Fig. 20

EFACE ID	FACE*	NEXT_EFACE*	NEXT_EDGE_ON_EFACE*
1	x	1, tile 2, 1	e5, tile 2, e2

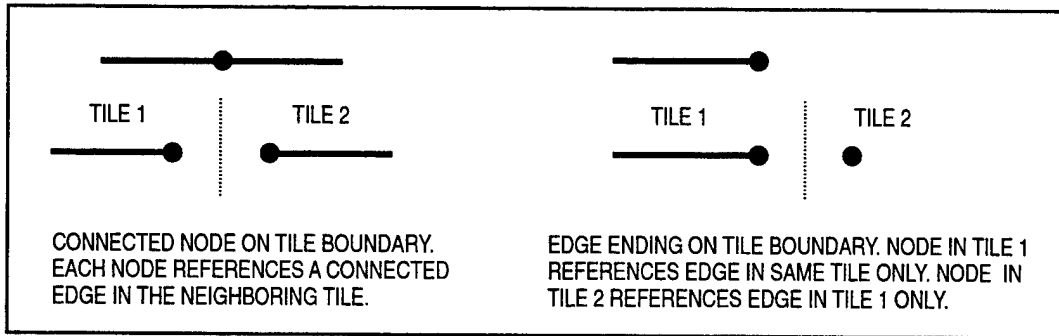*Note: Names rather than id's may be used for clarity.

Fig. 21 — Cross-tile connected node rules [1]

9.0 SPATIAL INDEXING

This section describes the Spatial Index File in VPF+. VPF provides spatial indexing as an optional means of improving query performance. An application can, for example, retrieve all primitives found within a bounding region without exhaustively searching all primitive tables. The VPF+ spatial index derives from a grid-based 3D binary tree. The Spatial Index File header and bin data records are extended to include coordinates based on Level 4's Minimum Bounding Box (MBB).

9.1 Spatial Index Creation

In traditional VPF, a spatial index is created by dividing a tile into 2D areas that become cells of a grid-based binary tree. The first split divides the area into right and left halves. The second split is top and bottom halves, then right and left again, and so on. Multidimensional binary search trees such as this were analyzed in [5]. The data structure was found to be efficient in handling point, line, and area spatial queries with the user specifying conditions for multiple keys (here, multiple coordinates). The traditional VPF method is, therefore, extended in VPF+ to a grid-based 3D binary tree, with cell splitting in a different dimension at each level of the tree.

The 3D binary tree is created as follows. The tile is divided into 3D regions that become cells of the tree. Each division splits the parent cell into half. The division is made by a planar division of the cell into two equal 3D regions. The dividing plane is perpendicular to and intersects one of the x, y, and z axes at alternating levels of the tree. The first split divides the entire tile into right and left 3D regions along the x axis. The second split is into rear and front 3D regions along the y axis. The third split is into top and bottom 3D regions along the z axis. The process is then repeated. Each 3D region resulting from the division becomes the right or left child of its parent cell. When a split occurs, the primitives are distributed down into either child cell depending on the primitive's MBB. A primitive remains with the parent cell only if its MBB intersects the boundary between a parent's children cell.

The process for dividing a tile is partially illustrated in Fig. 22. In Fig. 22, the plane dividing each cell is represented by bold-faced lines. Cells 2 and 3, the two children of cell 1, point to the primitives in the right and left 3D regions of cell 1 (the entire tile). Cells 4 and 5, the two children of cell 2, point to the primitives in the back and front 3D regions of cell 2 (the right-hand region of the entire tile). Cells 8 and 9, the two children of cell 4, point to the primitives in the top and bottom 3D regions of cell 4 (the right-rear region of the entire tile).
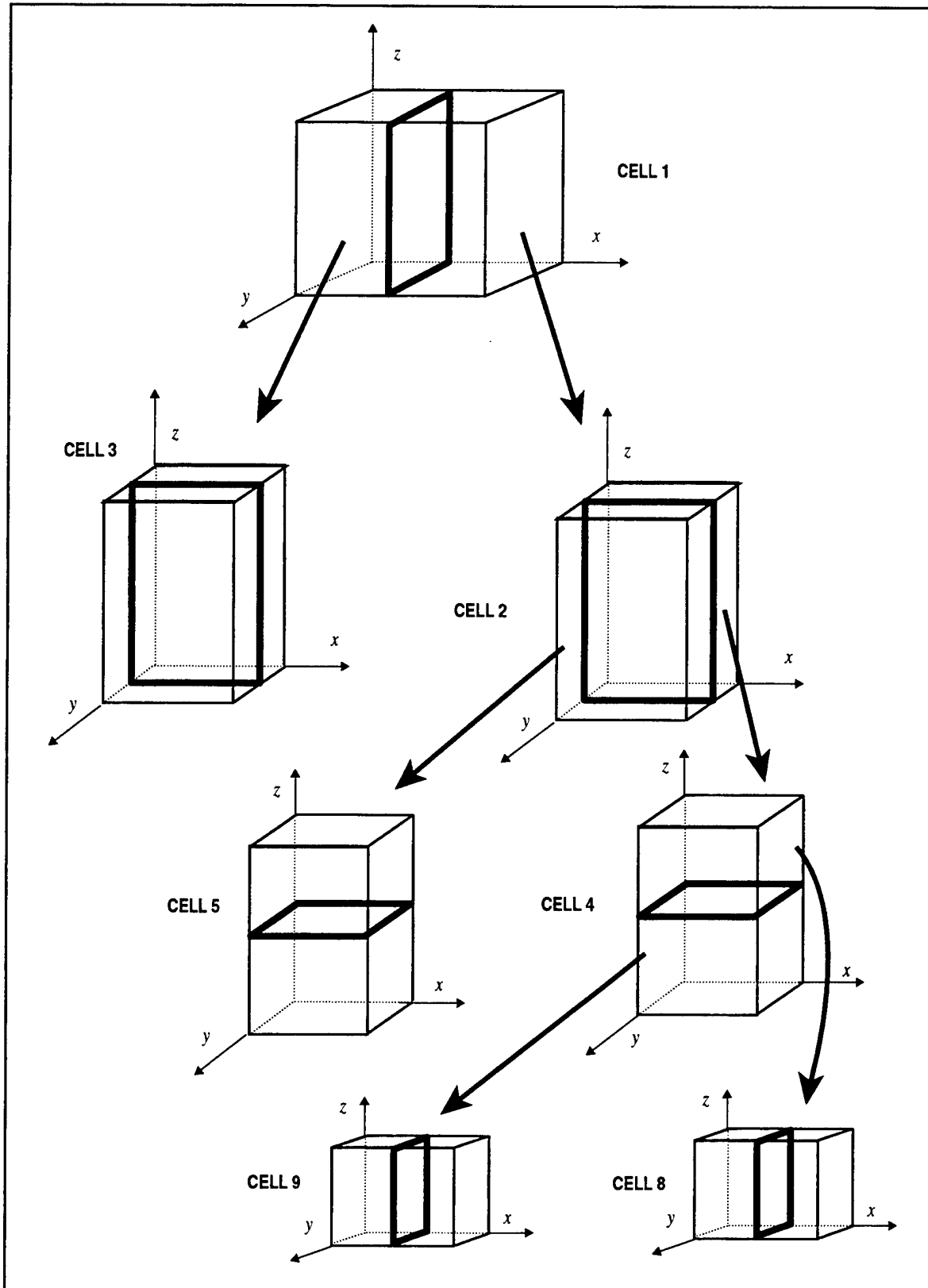
Fig. 22 — 3D spatial index cell decomposition

Each cell of the tree contains a list of primitive MBBs together with a list of the primitive id's that are found at that level of the tree. Cells are split when the number of primitives exceed a bucket size specified by the product specification. The criteria for splitting cells and for ending the splitting process in VPF+ remain unchanged from traditional VPF.

VPF uses a 1-byte integer spatial index coordinate system to which primitive coordinates are converted prior to indexing. The same conversion formula are utilized in VPF+ for minimum (x1,y1,z1) and maximum (x2,y2,z2) coordinates.

9.2 Spatial Index File Format

The format of the VPF+ Spatial Index File consists of a header, a bin array of the tree, and bin data records for each primitive in the tree. The header and data records files are extended to include the coordinates of each primitive's MBB. The bin array record remains unchanged from traditional VPF.

* *Header* – The header contains the number of primitives, the MBB of the entire spatial extent of the tree, and the number of cells of the tree. The spatial index file header record layout is described in Table 15.

* *Data Records* – There is one record for each primitive of the tree. Each record contains the coordinates defining the MBB for a primitive and that primitive's id. The structure of the data record is shown in Table 16.

10.0 RESULTS AND FUTURE WORK

This work has described the data structures and data organization of VPF+. VPF+ was designed to provide for georelational modeling in 3D and with the capability of modeling a wide range of objects that may be received or transmitted through SEDRIS.

The SEDRIS project, including the object model, its role in data interchange, etc., was examined and the class of objects that may be encountered in a SEDRIS transmittal was identified to include

Table 15 — Spatial Index File Header Record Layout

BYTE OFFSET	WIDTH	TYPE	DESCRIPTION
0	4	Integer	Number of primitives
4	4	Floating Point	MBB x1
8	4	Floating Point	MBB y1
12	4	Floating Point	MBB z1
16	4	Floating Point	MBB x2
20	4	Floating Point	MBB y2
24	4	Floating Point	MBB z2
28	4	Integer	Number of Cells in Tree

Table 16 — Structure of the Bin Data Record

BYTE OFFSET	WIDTH	TYPE	DESCRIPTION
HDR + BIN + OS + c * 8 + 0	1	byte	MBB x1
HDR + BIN + OS + c * 8 + 1	1	byte	MBB y1
HDR + BIN + OS + c * 8 + 2	1	byte	MBB z1
HDR + BIN + OS + c * 8 + 3	1	byte	MBB x2
HDR + BIN + OS + c * 8 + 4	1	byte	MBB y2
HDR + BIN + OS + c * 8 + 5	1	byte	MBB z2
HDR + BIN + OS + c * 8 + 6	4	int	Primitive id

Note: c is {0..(number of primitives for a cell − 1)}; the c value for the first primitive is 0. HDR is the length of the index file header record. BIN is the summed length of all the bin array records. OS is the value of the offset variable in the corresponding bin array record.

non-manifold as well as manifold objects. The winged-edge topology of VPF was found to be deficient to represent non-manifold objects. Several data structures capable of representing both of these classes of objects were therefore studied. These data structures were not directly implementable, but they did provide a theoretical basis for the data structure that was designed.

As a result of these investigations, a new, extended winged-edge data structure called non-manifold 3D winged-edge topology was designed for a new level of topology, Level 4 full 3D topology. A new structure called efaces was developed to resolve some of the ambiguities found in non-manifold representations. This topology was designed to be a logical extension to the winged-edge topology of traditional VPF, and no changes were made that alter traditional VPF's lower levels of topology.

On the feature level, a new 3D_Object Feature class was introduced to capture a wide range of 3D objects. These features were further defined to be either Well Formed or Not Well Formed, with Well Formed 3D_Object Features having additional optional topologic information to improve software performance.

Further investigation is continuing in several areas. First, and most obvious, are issues related to data storage and to data structure efficiency. Second, since a synthetic environment should be representable at various levels of resolution depending on the needs of the user, the implementation of multi-scale surface and 3D object representation is being explored. Finally, how spatial tiling should be addressed in VPF+ is at issue in light of the SEDRIS method of spatial organization. In SEDRIS, tiles may be subdivided into smaller tiles. In traditional VPF, an entire coverage is tiled according to a single tiling scheme and tiles may not be divided into smaller tiles. Implementation of the SEDRIS spatial organizational scheme in VPF+ would make consumption of SEDRIS data straightforward, but would constitute an extreme departure from the structural organization of traditional VPF. Feedback is being sought from the VPF community on the benefits and disadvantages of each approach.

11.0 ACKNOWLEDGMENTS

12.0 REFERENCES

1. Department of Defense, "Interface Standard for Vector Product Format," MIL-STD-2407, 28 June 1996.

2. Woo, T. C., "A Combinatorial Analysis of Boundary Data Structure Schemata," *IEEE Computer Graphics and Applications* **5**(3), 19–27 (1985).

3. Abdelguerfi, M., E. Cooper, C. Wynne, and K. Shaw, "An Extended Vector Product Format (EVPF) Suitable for the Representation of Three-Dimensional Elevation in Terrain Databases," *International Journal of Geographical Information Systems* **11**(7), 649–676 (1997).

4. O'Rourke, J., "Computational Geometry in C," Cambridge University Press, 1995.

5. Bently, J. L., "Multidimensional Binary Search Trees Used for Associated Searching," *Commun. ACM* **18**(9), 509–517 (1975).

Appendix A

REPRESENTING NON-MANIFOLD GEOMETRIC MODELS

Three boundary-based data structures capable of modeling non-manifold objects are described below. The three data structures to be described are the Radial Edge [2], the Tricyclic Cusp [1], and the ACIS Geometric Modeler [3].

1.0 THE RADIAL EDGE STRUCTURE

The Radial Edge Structure [2] is an edge-based data structure that addresses topological ambiguities found with two non-manifold situations—the non-manifold edge and the non-manifold vertex. Generally, this is accomplished through a radial ordered list of faces connected along a common edge and a radial unordered list of edges connected at a given vertex. Additionally, separate structures defining the "use" of faces, loops, edges, and vertices are used to represent adjacency relationships rather than the faces, loops, edges, and vertices themselves.

There are several topological restrictions on the domain of objects that can be represented by the Radial Edge Structure. Among these, while a surface is allowed to be non-manifold, individual faces are required to be manifold. This prevents a face from self-intersecting except along its boundary. Additionally, edges may only intersect faces along or at their boundaries. Generally, any two topological elements may intersect each other only at a level of hierarchy at least one level down from the lowest of the two intersecting levels (i.e., different regions may intersect each other only along their boundaries).

1.1 Topological Elements

The topological elements used by the Radial Edge Structure are organized in the following order from top to bottom: model, region, shell, face, loop, edge, and vertex. Lower dimensional elements in the hierarchy serve as boundaries for higher dimensional elements.

A model can be thought of as a 3D modeling space that holds all elements contained in a geometric model. A model is technically not a topological element, but more of a bucket to hold all of the topological elements of a geometric model. There is at least one region in a model. A region is a volume of space. For example, a single cube requires two regions, one for the inside and the other for the outside. Only the latter region would have an infinite extent. When there is more than one region, all have a boundary.

A shell consists of an oriented boundary surface. It may be connected to form a closed volume or it may be an open set of adjacent faces, a wireframe, a combination of faces and wireframe, or a single point. A solid with an internal void would have more than one shell. No shell exists if all

31

space exists as a single region. In that case, no modeling has yet been done or all models have been deleted.

A face is the bounded portion of a shell excluding the boundary. Though a face is orientable, it is not oriented since two region boundaries may use different sides of the same face. A loop is a connected boundary of a single face. This can be either a single vertex or alternating sequences of edges and vertexes. A face can have more than one loop (a face with a hole in it). Loops are not oriented as they bound a face that may be used by two different shells. The use of a loop is oriented. An edge is bounded by vertex at each end, possibly the same one. It is orientable, but not oriented. The use of the edge, on the other hand, is oriented. A vertex is simply defined as a unique point in space.

The Radial Edge Structure also defines face use, loop use, edge use, and vertex use as additional topological adjacency elements associated with faces, loops, edges, and vertices. As mentioned, these "uses" are the elements employed to represent adjacencies. Although this increases storage costs, it simplifies traversing the structure by providing through the "usage" a unique identification for each face, loop, edge, and vertex usage.

Face use represents the use of one of the two sides of a face by a shell and is oriented with respect to face geometry. Loop use represents one of the uses of a loop associated with one of the two uses of a face and is also oriented with respect to its associated face use. Edge use is the use of an edge by a loop use. There is one edge use for each face side. Orientation is specified with respect to edge geometry. Finally, a vertex use can represent the use of a vertex by an edge as an endpoint. There can be a single vertex loop or a single vertex shell. In those cases, vertex use represents use of the vertex by the loop or shell, respectively.

1.2 Data Structure Relationships

The Radial Edge Data Structure relationships are shown in Fig. A1. Generally, the root of the data structure is a list of all models, with a pointer to the next and last models in the list and to a list of regions in the modeling space. Each region has pointers to its parent model, to the next and last regions in the list of regions, and to one node of a list of shells in the region. Each shell points to its parent region, to the next and last shell in the list of shells, and to one of three
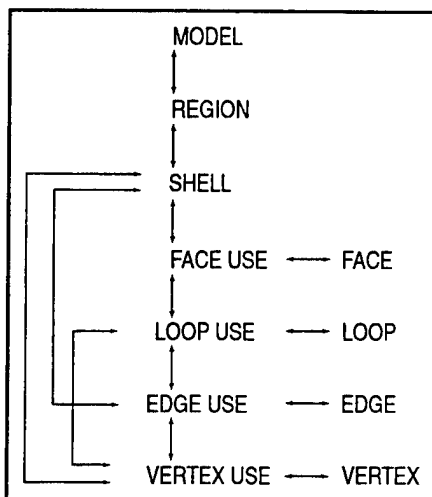

Fig. A1 — Radial Edge Structure relationships [2]

alternative topological adjacency use elements, face use, edge use, or vertex use. Faces, loops, edges and vertices point to their own corresponding "uses" and to their own attributes, including geometry.

In addition to having pointers to their corresponding faces, loops, edges, and vertices, the face use, loop use, edge use, and vertex use structures have additional pointers necessary to maintain the adjacency relationships. In particular, the face use structure contains pointers to its parent shell, as well as the next and last face uses in the shell's list of face uses. It also contains a mate pointer to the opposite side of the face and a pointer to a list of loop uses.

The loop use structure has pointers to its face use parent, to the next and last loop use in the face use's list of loop uses, to the loop use on the other side of the face, and one pointer to a lower level in the hierarchy. That pointer is to a vertex use when the loop is comprised of only one vertex; otherwise, it is to a vertex in a list of edge uses in the loop use.

The edge use has pointers to the starting vertex use of the edge use in the current orientation, to the edge use on the other face use of the face, and one pointer to a higher level in the hierarchy. That pointer is either (1) to its parent shell or (2) to the next and last edge use in an ordered edge use list, to the edge use on the radially adjacent face use, and to its loop parent.

The vertex use has pointers to the next and last vertex use in a list of all vertex uses of the vertex. It also has a pointer to one of the following: (1) its parent shell when there are no face uses or edge uses on the shell, (2) its parent loop use when the loop consists only of the current vertex use, or (3) the edge use causing the current vertex use.

1.3 Illustrations

Some of the face use, edge use, and vertex use relationships are illustrated in Figs. A2, A3, and A4. Figure A2 shows a cross-section of three faces (f_1, f_2, f_3) sharing a common edge (e_1). Since each face has two face use structures, one for each side of each face, there are a total of six face use structures represented by fu_1, fu_2,..., fu_6. Each face use points to the face use on the opposite side of its own face. There are six edge use structures (eu_1, eu_2,..., eu_6), one for each of the two sides of each of the three faces. Each edge use points to the edge use on the other side of the face and to the edge use structure on the face use radially adjacent to itself. For example, in Fig. A2, eu_1 points to eu_2 and to eu_6.

Figure A3 shows a vertex (v_1) with five connected edges. Each of the five edges has an edge use (eu_1, eu_2,..., eu_5, for example). Since there are five edges, the vertex has five vertex uses (vu_1, vu_2,..., vu_5). The vertex points to one of its vertex uses (v_1 points to vu_5 through vvuptr). Each vertex use points to its neighboring vertex use (vu_5 points to vu_1 and to vu_4, for example) and points to vertex v_1 (through vuvptr). Finally, each vertex use points to its connected edge use through the "vueuptr" pointer (vu_1 points to eu_1) and each edge use points to its vertex use through "euvuptr" (eu_1 points to vu_1).

Fig. A4 shows the relationships involved in a loop of edge uses. The loop is given as a loop use (lu_1). Since there are four vertices and four edges, there are four vertex uses (vu_1 through vu_4) and four edge uses (eu_1 through eu_4). Each vertex use points to the edge use causing the vertex use (e.g., vu_1 points to eu_2). Each edge use points to its corresponding vertex use (e.g., eu_2 points to vu_1), to its parent loop (lu_1), and to the next and last edge use (e.g., eu_1 points to eu_4 and to eu_2) through "lueulast" and "lueunext."
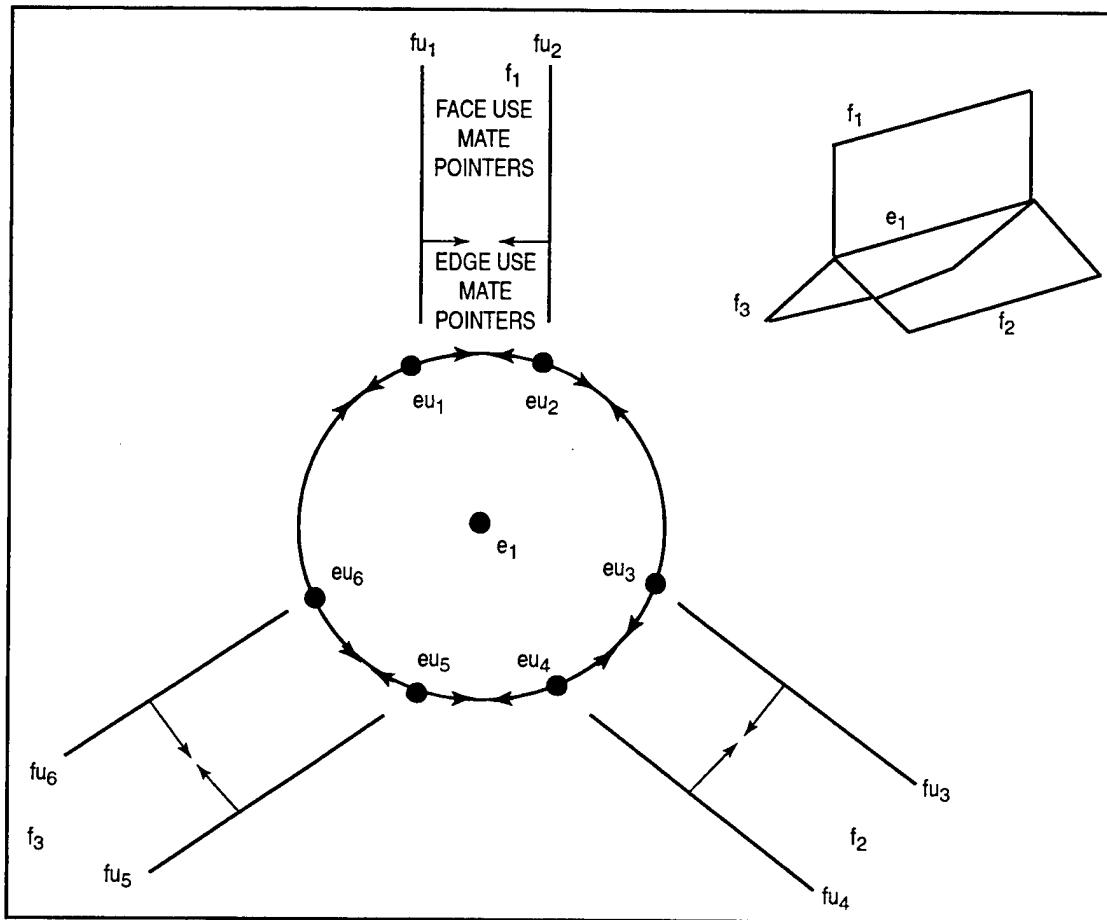
Fig. A2 — Faces sharing an edge in the Radial Edge Structure [2]

2.0 THE TRICYCLIC CUSP STRUCTURE

The Tricyclic Cusp Structure [1] is a vertex-based data structure. This data structure addresses the topological relationships that the Radial Edge Structure addresses, and in addition, is specifically intended to resolve ambiguities inherent in certain non-manifold representations that may not be easily eliminated by the Radial Edge Structure. An example is the situation depicted in Fig. A5 in which two open cones (A and B) are joined at vertex "v." Vertex v is not connected to any edges. Traversal from the outside of cone A to the outside of cone B may not be guaranteed with the Radial Edge Structure.

2.1 Topological Elements and Data Structure Relationships

The topological elements of the Tricyclic Cusp Structure are the model, region, face, edge, vertex, shell, wall, edge orientation, zone, disk, loop, and cusp. The relations between these elements are shown in Fig. A6.

The vertex, edge, face, shell, and region have meanings similar to their counterparts in the Radial Edge Structure. A wall is equivalent to the Radial Edge Structure's "face use"—one of
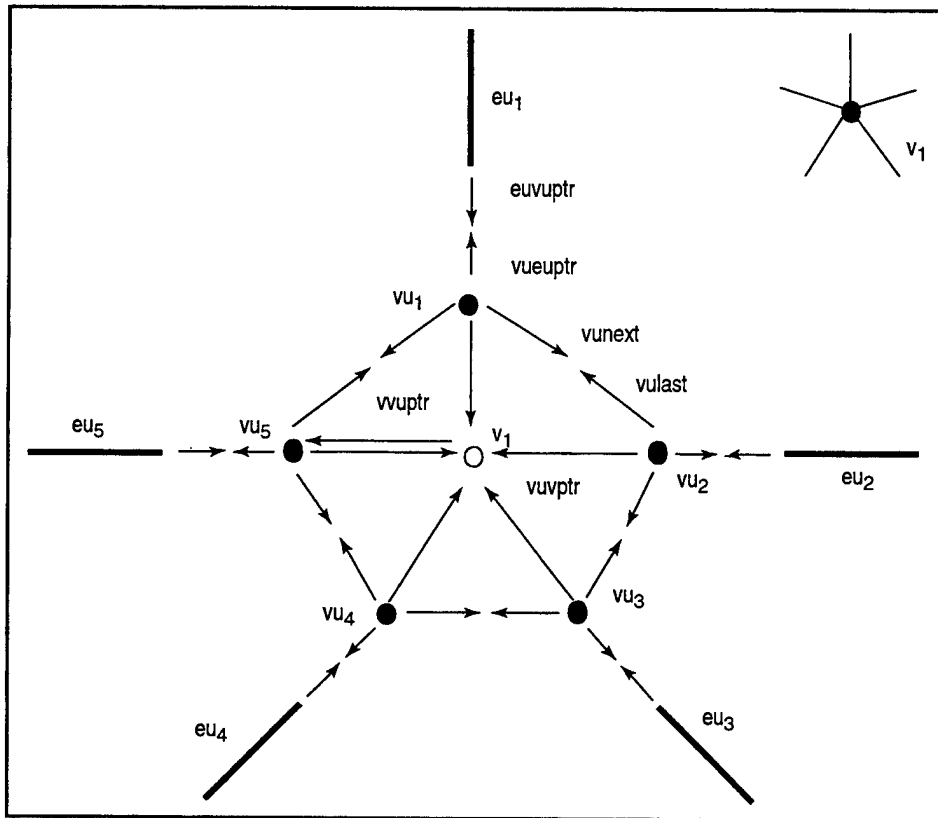
Fig. A3 — Edges sharing a vertex in the Radial Edge Structure [2]
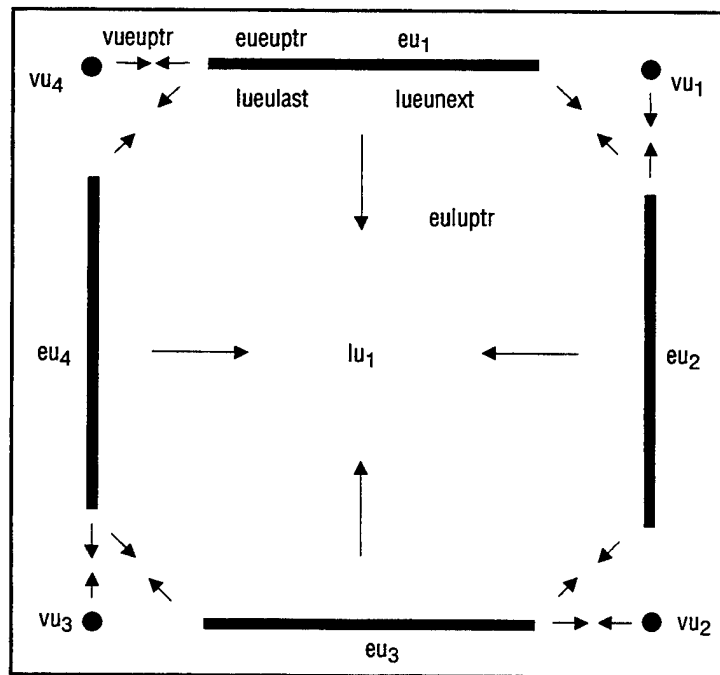
Fig. A4 — Loop of edges in the Radial Edge Structure [2]
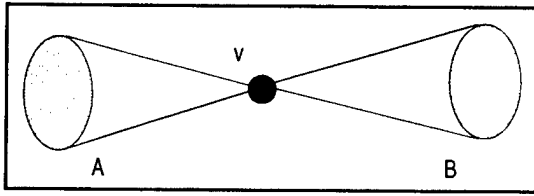
Fig. A 5 — Representation rendered ambiguous in Radial Edge Structure [1]
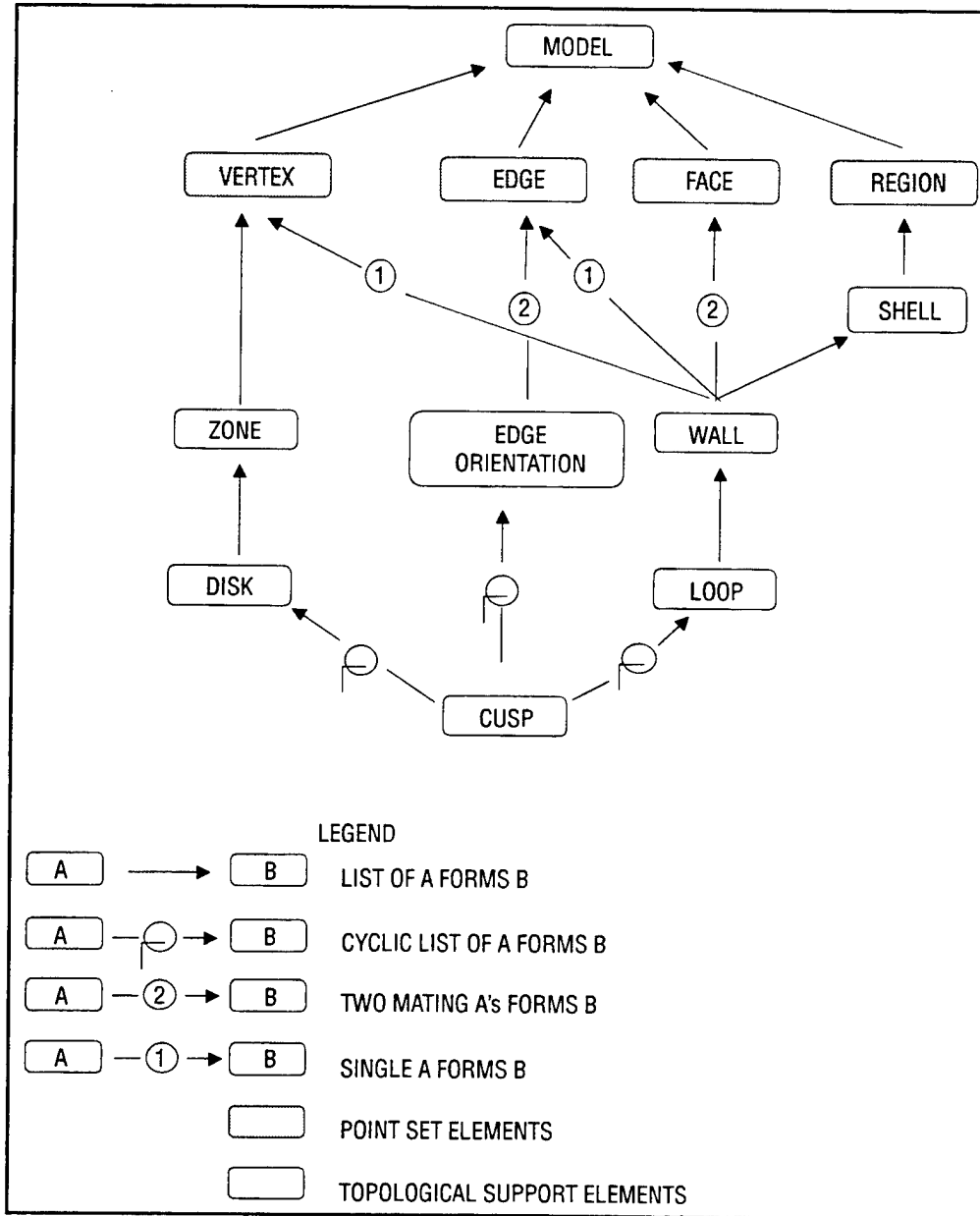
Fig. A6 — Tricyclic cusp data structure [1]

the two oriented sides of a face (Fig. A7). The ordering of adjacent vertices defines an edge orientation. There are two possible orientations for each edge.

The remaining topological elements can most easily be understood by starting with the cusp (Fig. A6). This term is used as an "entity" that is the representation of enhanced vertex use information or is descriptive of the neighborhood around a vertex. It can be thought of as the fusion of vertex use and edge use information. Cusps are depicted in Fig. A7. As shown in Fig. A7, a cyclic ordering of cusps about a wall forms a loop. The ordering is consistent with wall orientation. A loop bounding a face would have a mate loop—the wall on the opposite side of the face, called the mate wall. The mate cusp, also shown in Fig. A7, relates to the mate wall and same vertex.

Three circular lists of cusps form the edge orientation, disk, and loop (Fig. A6). Figure A8 shows the circular lists related to the edge orientations for three faces connected along one edge. An ordered list of all cusps associated with the first vertex indicated by the edge orientation make up one edge orientation cycle forming an edge. The second edge orientation is made up of the cusps associated with the vertex indicated by the opposite edge orientation. The mate edge orientation relationship is used in connection with the two ordered lists to locate each face connected at the common edge. As noted in Fig. A8, two possible edge orientations are associated with each "real" edge. Both oriented walls of a face point to the "real" edge, but an unoriented wall points to an isolated vertex.

Zones and disks are utilized to elaborate the vertex uses around a given vertex. A zone is a 3D region around a vertex bounded by a disk. In Fig. A9, a disk cycle is represented. The cycle of cusps is for the vertex of the cube as seen from the outside and simulates one side of an open disk. Each vertex has one or more zones and each zone has one or more disks. A shell is composed of
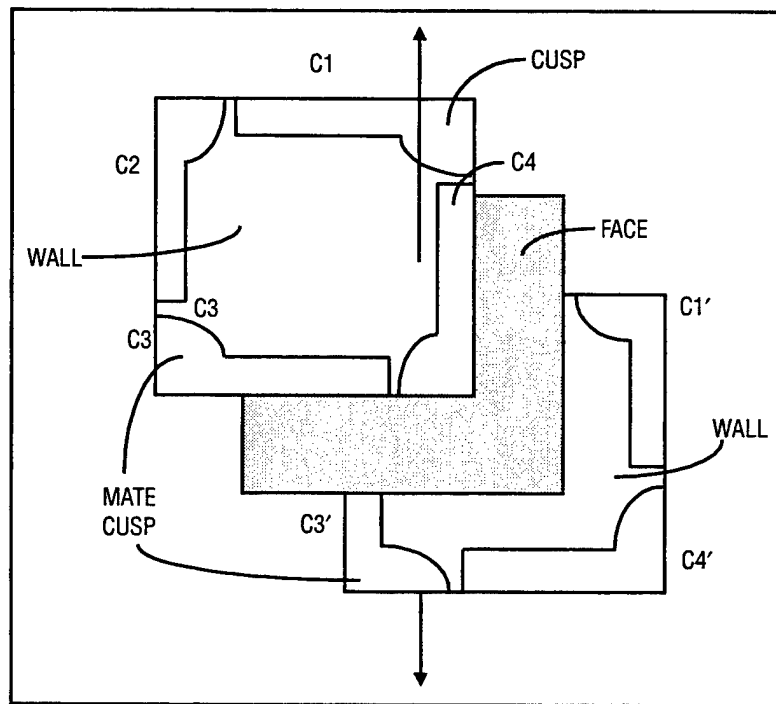

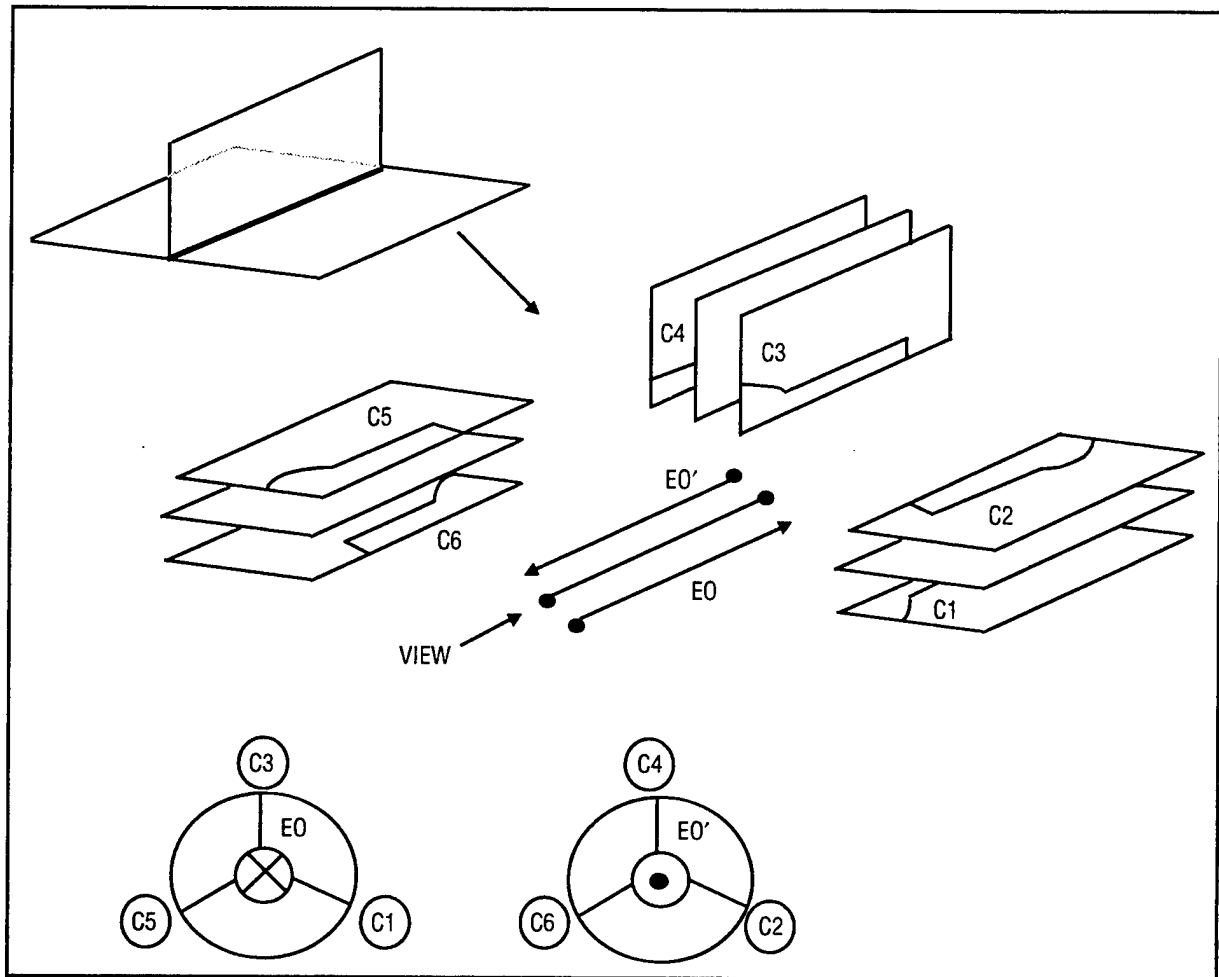Fig. A7 — Cusp, face, and wall description [4]

Fig. A8 — Edge orientation cycle in the Tricyclic Cusp Structure [1]

a list of walls, and a region is made up of a list of shells. Finally, a model serves to incorporate all elements of an object and more than one model can exist.

3.0 THE ACIS GEOMETRIC MODELER

The ACIS Geometric Modeler [3,5] is a component-based package consisting of a kernel and various application-based software components. The topology-related data structures are found in the kernel. The ACIS Geometric Modeler can model 1D, 2D, and 3D objects, including those with non-manifold edges and non-manifold vertices.

3.1 Topological Elements

The hierarchy of elements comprising ACIS topology are shown in Fig. A10. At the highest level, a body models a solid object, which can consist of more than one disjointed body treated as one. Next is the lump, which represents a connected portion of space bounded by one or more shells. A shell can refer to the boundary of a void internal to an object or can refer to the peripheral
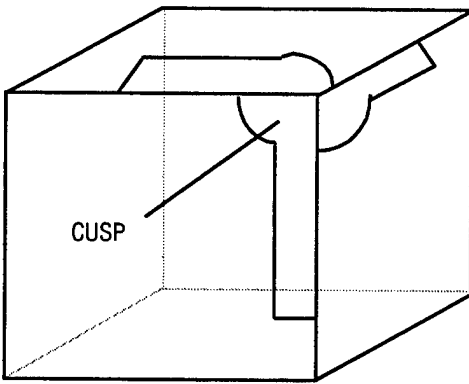
Fig. A9 — Disk cycle forming a disk around a vertex [1]
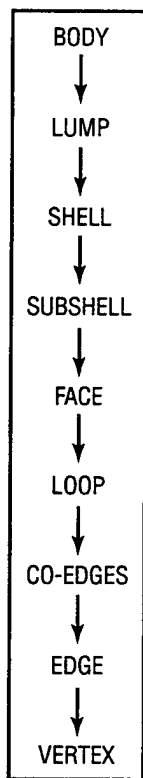
Fig. A10 — ACIS topology hierarchy [5]

boundary of a lump. Shells are made up of a collection of faces, and shells can be subdivided into subshells. A face is a surface bounded by loops of edges. A face can be single- or double-sided. When single-sided, the points on one side of a face are considered inside of the shell and the points on the other side are on the outside. When double-sided, all points on either side of the face are either all inside or all outside. Loops are connected portions of the boundary of a face and consist of a list of co-edges.

Co-edges in the ACIS Geometric Modeler are comparable to, but not exactly like, edge uses in the Radial Edge Structure and cusps in the Tricyclic Cusp Structure. In addition to relating edges with adjacent edges, co-edges make it possible for edges to occur in more than one adjacent face. This is because each edge has one co-edge for each adjacent face (Fig. A11). In Fig. A11, three faces of a solid are shown. The line with arrows indicate the loops of co-edges for each face.

Next, edges are represented as physical edges bounded by two vertices. If either vertex is null, then the edge is unbounded in that direction, and if both vertices are identical, then the edge is considered an isolated point, such as the apex of a cone.

3.2 Data Structure Relationships

In the ACIS Geometric Modeler, the body contains a pointer to the first lump's shell in the body. The lump points to the next lump in the body, the first shell in the lump, and the body that contains the lump. The shell has pointers to the next shell in the lump, the first subshell, if any, the first face in the shell, and the lump containing the shell.

The relationships of the faces, loops, co-edges, edges, and vertices are shown in Fig. A12. References to geometric information are omitted. Generally, each face points to the next face in the shell, the first loop bounding the face, and the shell containing the face. Each loop points to the face that the loop bounds, the next loop in the face boundary, and the first co-edge in the loop. The co-edges point to the next and previous co-edges in the loop, the partner co-edge on the edge (to allow for more than two faces connected along a common edge), the edge on which the co-edge lies, the co-edge direction, and the owning loop. Co-edge direction is significant because the co-edges are oriented such that the face is always on the left when the observer is looking along the co-edge with the outward pointing face normal upward. Each edge points to its starting and ending vertex and the first co-edge on the edge. Finally, a vertex points to the edge using the vertex. In the case of two
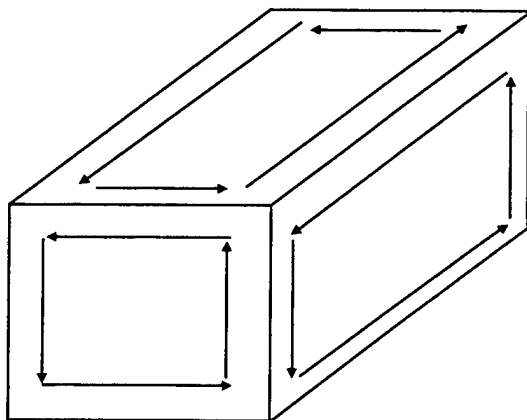
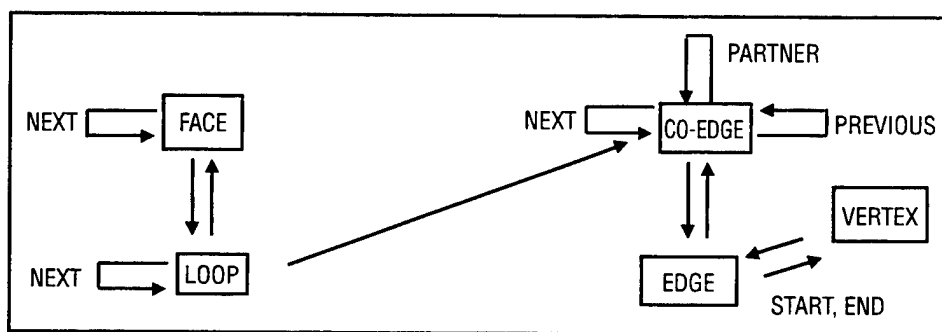Fig. A11 — Co-edges in loops [5]

Fig. A12 — Some of the Data Structure Relationships in ACIS [5]

non-manifold situations (two objects connected only at a single vertex or a dangling edge), the vertex can point to the multiple edges to allow access to all edges at the vertex.

4.0 REFERENCES

1. Gursoz, E. L., Y. Choi, and F. B. Prinz, "Vertex-Based Representation of Non-Manifold Boundaries," Geometric Modeling for Product Engineering, Elsevier Science Publishers B.V. (North-Holland), 1990.

2. Weiler, K. J., "Topological Structures for Geometric Modeling," Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, 1986.

3. Spatial Technology, Inc., *Format Manual*, 2425 55th Street, Building A, Boulder, CO, 80301, 1996.

4. Leonidas, B. and N. Patrikalakis, "Topological Structures for Generalized Boundary Representations," Sea Grant College Program, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1994.

5. Spatial Technology, Inc., *Applications Guide*, 2425 55th Street, Building A, Boulder, CO, 80301, 1996.

Appendix B

SEDRIS OVERVIEW

This appendix is organized as follows. Sec. 1.0 introduces the SEDRIS project and covers the portions of the SEDRIS object model relevant to this research. Section 2.0 covers significant differences between VPF and SEDRIS. All references to the SEDRIS object model are to [1] and all references to VPF are to [2].

1.0 INTRODUCTION

The Synthetic Environment Data Representation & Interchange Specification (SEDRIS) is a project funded by the Defense Modeling and Simulation Office for the development of a standard mechanism for interchanging synthetic environment data. "Synthetic Environment Data" includes a diversity of digital data used to represent varied aspects of the real world environment. Some of the data that can be considered synthetic environment data includes data about the terrain, ocean, atmosphere, natural and man-made models, features, and associated sounds. As an interchange mechanism, SEDRIS is intended to allow a user to access, extract, and interpret this data exactly as a producer intended without changes to the data that would alter the characteristics of the information as intended by the producer. The SEDRIS project includes the development of a format independent data model, a data format, and an Application Program Interface (API) [3]. The data model should become stable in the near future once issues involving the representation of topology, color, atmospheric, and oceanic data have been resolved. Though an interim data format has recently been suggested, a final format will probably await stability of the data model. Since the SEDRIS object model is not yet stable and there is no SEDRIS format, the API currently provides for "read only," allowing conversion only between a native data base model and the SEDRIS object model held in RAM. Once the SEDRIS project is complete, the API will enable conversion between major native data base models and formats and the SEDRIS model and format.

1.1 The Data Model

The SEDRIS data model is an object-oriented data representation model [1]. The model is generally organized around five classes: Synthetic_Environment, Features, Geometry, Feature_Topology, and Geometry_Topology. These five classes and related subclasses illustrate the general organization and structure of SEDRIS and also provide a foundation for the comparison of SEDRIS to other models such as VPF [2]. The entire SEDRIS model is not presented in detail here. The omitted portions are not considered significant to an overall understanding of the structure of the SEDRIS object model and its relationship to the VPF relational model. All diagrams use the modified Raumbaugh notation employed by SEDRIS, which is explained in App. C.

41

1.2 Synthetic_Environment

The Synthetic_Environment class is the minimum object required in a SEDRIS transmittal and is the object around which the data base is organized. Figure B1 shows some of the relationships from Synthetic_Environment. As shown in Fig. B1, a Synthetic_Environment has zero or more Color_Tables, at most one Base, Sound_Library, Texture_Library, Symbols_Library, and Model_Library, and exactly one Transmittal_Encoding and A&L (Accuracy and Lineage).

Transmittal_Encoding and A&L are conceptually grouped as subclasses of the abstract Metadata class because each lets the user know something about the data contained in the transmittal. Transmittal_Encoding, for example, might identify those parts of the Synthetic_Environment that are and are not available. A&L, on the other hand, might include such information as the source of the data and date of last modification.

The libraries are optional, but if they exist, they organize all of the sounds, textures, symbols, and models that can be instanced in the synthetic environment. A model may be something as simple as a generic building that will be instanced multiple times in a single data base or something as complex as a VPF coverage.

The base object shown in Fig. B1 organizes the Feature and Geometry data that is not found under model. A SEDRIS feature generally corresponds to traditional VPF-type data, while geometry is used to describe things that have substance.
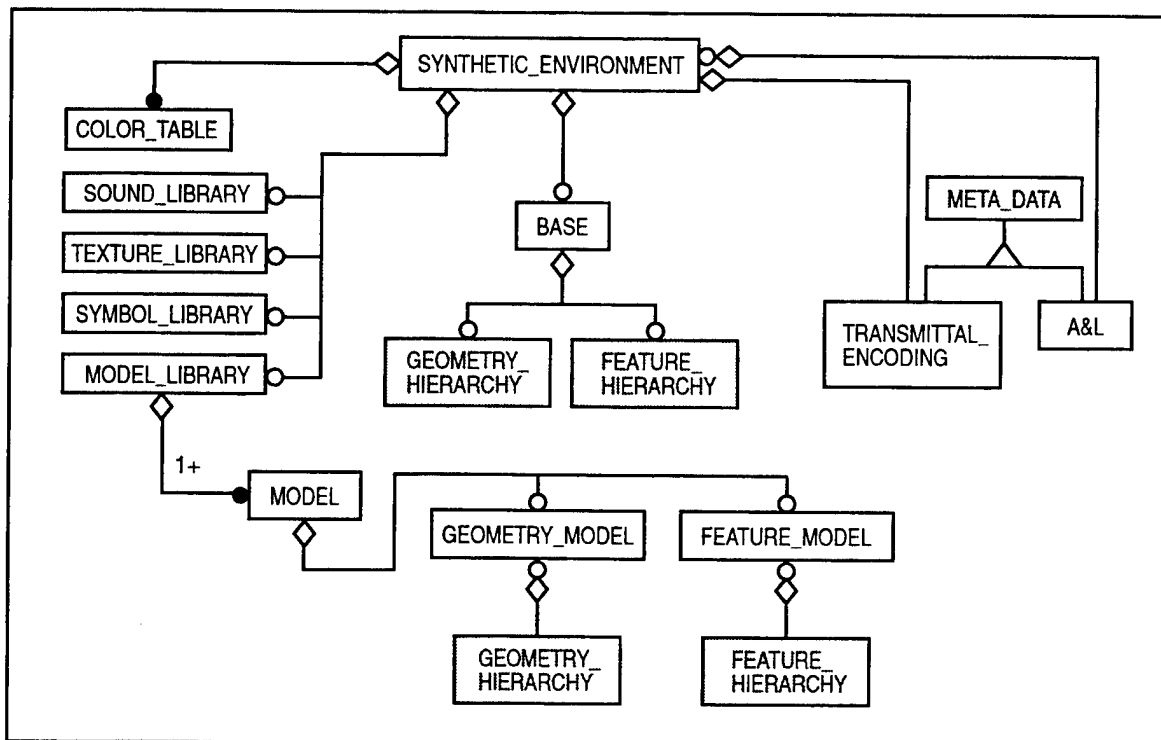
Fig. B1 — Synthetic environment and related classes [1]

1.3 Features

A feature is used to conceptually organize "feature data," and as with all abstract classes, it is never instanced. Figure B2 depicts the feature class along with some of its descendants and relationships. A feature can be related to other features and to Geometry_Hierarchies. A feature is either a Primitive_Feature or Feature_Hierarchy. The basic data that describes a feature is organized in the Primitive_Feature abstract class as either a Point_Feature, Linear_Feature, or Areal_Feature. These are defined by the SEDRIS data dictionary as follows:

- Point_Feature – a point referencing a cultural feature;

- Linear_Feature – a feature located along an edge;

- Areal_Feature – feature located in a bounded area.

As Fig. B2 shows, each of the feature primitives is related to its topological counterpart: Feature_Node, Feature_Edge, and Feature_Face classes.

A Feature_Hierarchy is either a Feature_Connection (Fig. B2) or an Aggregate_Feature (Fig. B3). The Feature_Connection class relates a particular instance of a feature model to the Feature_Model class (Fig. B1). This is an example of a one-way relationship in which the Feature_Model_Instance knows the model it is related to, but the model does not know what Feature_Model_Instances may be related to it.

The Aggregate_Feature class shown in Fig. B3 provides 10 concrete subclasses within which feature data can be grouped. For example, a spatial tiling scheme could be implemented through the Spatial_Index_Related_Features class or multiple levels of resolution could be grouped in the
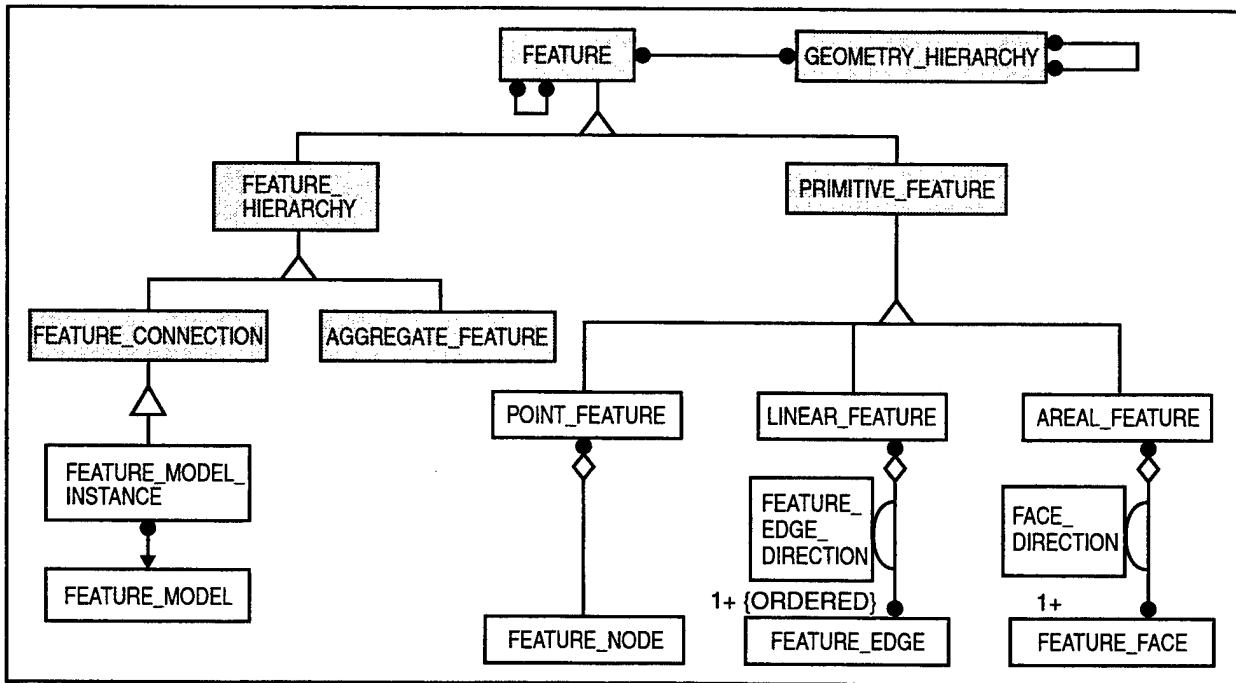

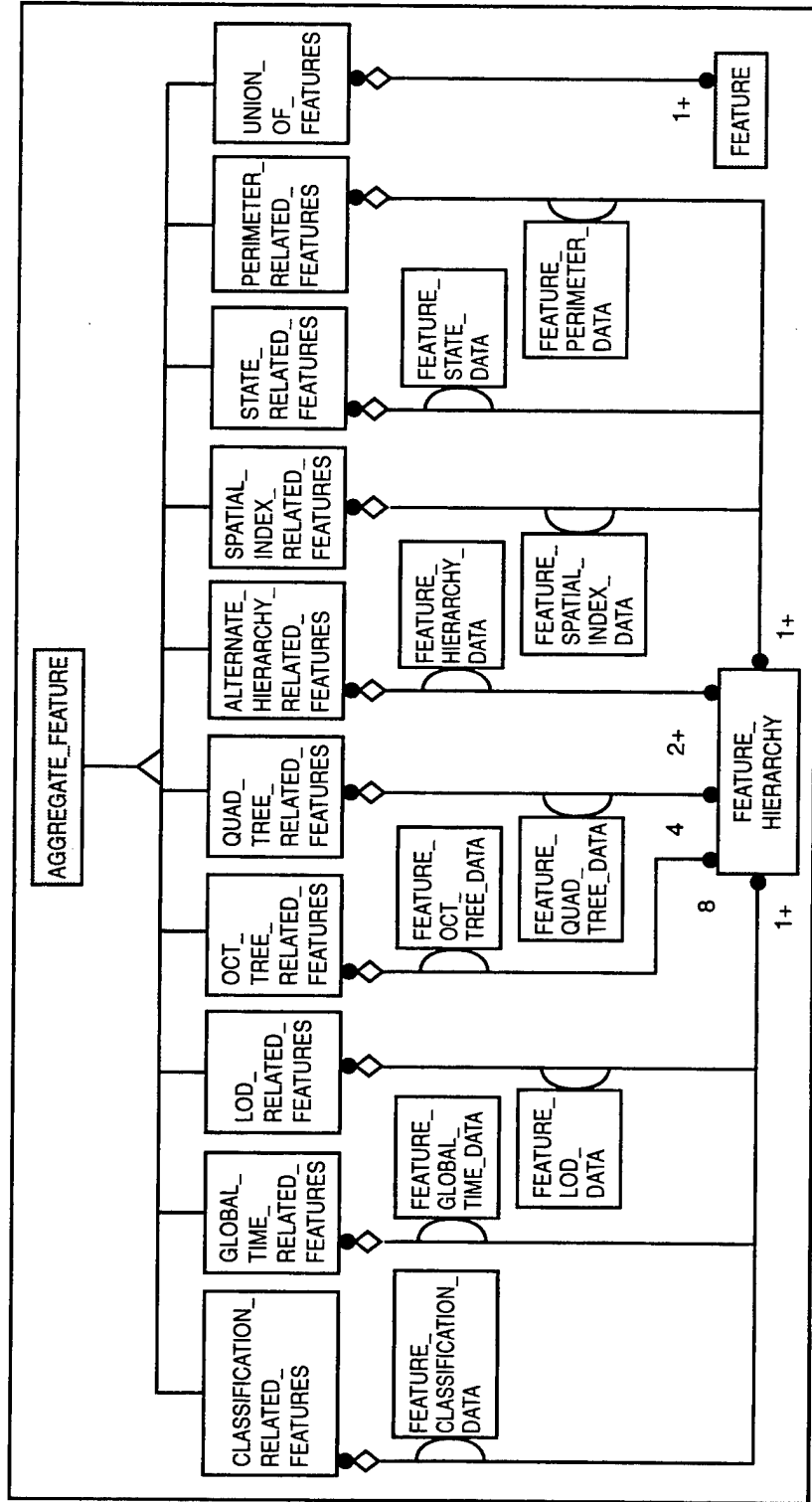Fig. B2 — Abstract feature classes, feature primitives, and feature model [1]

Fig. B3 — Aggregate features [1]

Level_of_Detail_Related_Features class. There can also be a mixture of different classes at different levels of the Feature_Hierarchy. For example, a Feature_Hierarchy can be composed of multiple levels of Aggregate_Features with each Aggregate_Feature grouping in the hierarchy consisting of a Level_of_Detail_Related_Feature. Alternatively, one Aggregate_Feature grouping in the hierarchy can consist of a Level_of_Detail_Related_Feature, and another grouping at a different level of the hierarchy can consist of Quad_Tree_Related_Features. Additionally, two different Aggregate_Feature class groupings can exist at the same level of the Feature_Hierarchy through Union_of_Features.

1.4 Geometry

The abstract geometry class and subclasses are shown in Fig. B4. A geometry is either a Primitive_Geometry or a Geometry_Hierarchy. Geometry_Hierarchies can be related to other Geometry_Hierarchies and to features.

1.5 Primitive Geometry

The Primitive_Geometry class specifies the basic data necessary to describe "geometry" objects. This type of data is divided into four classes defined by the SEDRIS data dictionary as follows:

* Linear_Geometry – geometric linear representations;

* Surface_Geometry – geometric surface representations;

* Volume_Geometry – volumetric representations;

* Point_Geometry – a specialization of primitive geometry in which all components have a Location_3D as a component, such as a light source or camera point.
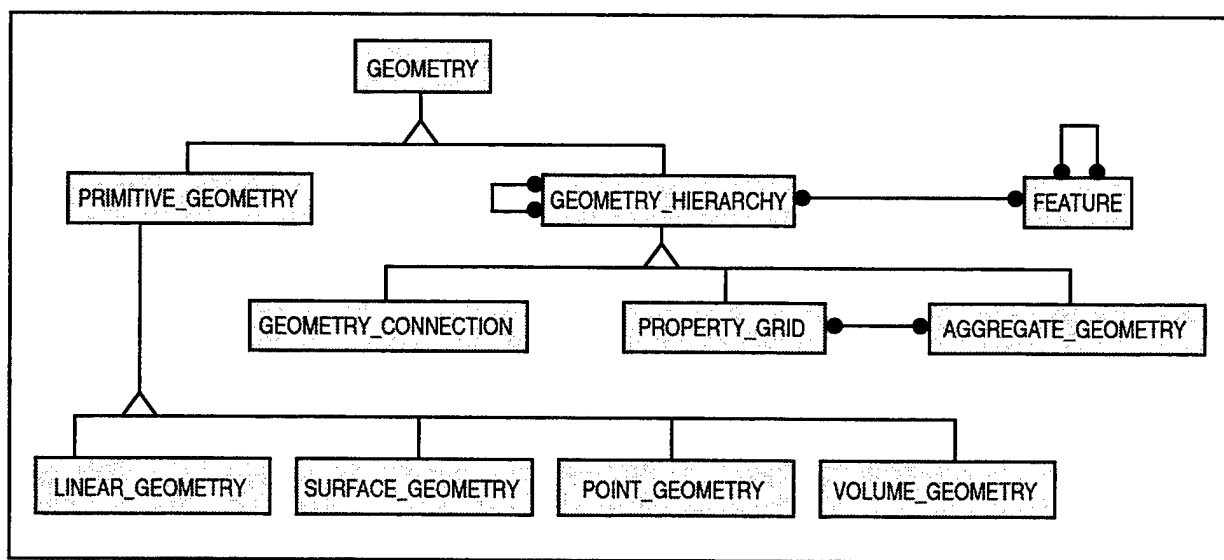

Fig. B4 — Abstract geometry class and subclasses [1]

Volume_Geometry may typically be utilized in oceanographic and possibly atmospheric applications. For example, a cold water eddy (in a sonar simulator application) would be represented as a "volume." Point_Geometry is not significant to this discussion and is omitted from further discussion.

Figure B5 shows the subclasses and interrelationships of Linear_Geometry and Surface_Geometry. The core primitives are line, vertex, and polygon or patch. The following points are noteworthy. A line is defined in SEDRIS as a set of ordered vertices. A vertex is the end of a line segment or the point of intersection of two polygons. A polygon differs from a patch in that the former is the bounded portion of a plane while the latter is a manifold defined by a parametric equation. The SEDRIS data dictionary gives an example of a patch to be the hood of a car. The normal can be used to determine the slope or orientation around an area.

The remainder of the geometric information is attached to vertex, and with the exception of Morph_Point, consists of primitive data. A Morph_Point is a location in 3D space significant to the transition of an object when morphing between levels of detail. For example, at one distance a building roof may look flat, but transition to the appearance of a peaked roof at a closer distance. Location_3D is the coordinate location of the vertex in 3D space. Location_3D is required while all other geometric information is optional. The conformal class indicates how a point will be forced to conform to another object. The example given in the SEDRIS data dictionary is that a conformal will determine whether a building floor will be coplanar with the surface below. Color and Color_Coefficient relate to the color and operations on that color. Texture_Coordinate maps a texel from an image to a vertex.

Unlike feature-related topology, geometry-related topology is optional in SEDRIS. This topo-logic information should be derivable from the geometric primitives. Line topology can optionally be found in the Geometry_Edge class, vertex topology in the Geometry_Node class, and polygon or patch topology in the Geometry_Face class. Topology is more fully discussed in a separate section.

1.6 Geometry_Hierarchy

Geometry_Hierarchy (Fig. B4) serves two functions. It captures associations with features. If there is an association with a feature, the geometry side can be considered an alternate represen-tation of the feature. Additionally, Geometry_Hierarchy conceptually organizes geometric data as either Aggregate_Geometry, Property_Grids, or Geometry_Connection.

Figure B6 shows the subclasses and relationships of Aggregate_Geometry. As with the Aggregate_Feature class discussed above, each of the subclasses provides a different method of grouping geometric data, and different methods can be found at different levels in the Geometry_Hierarchy.

Property_Grids are shown in Fig. B7. There can be one-, two- or three-dimensional grids. Grid_1D is a 1D matrix of grid vertices and references such primitive data as spacing units (meters, seconds, millibars). Grid_2D is a 2D matrix of evenly spaced vertices used to store such properties as elevation. Grid_3D is a 3D matrix adding "z" spacing such as an array describing a 3D volume of a bounded region. Feature_Node, shown in Fig. B7, relates to topology, described below.

Details of the transformation class are not shown in Fig. B7. This class organizes the data necessary for the rotation, scale, or translation to be applied to an object. It also holds
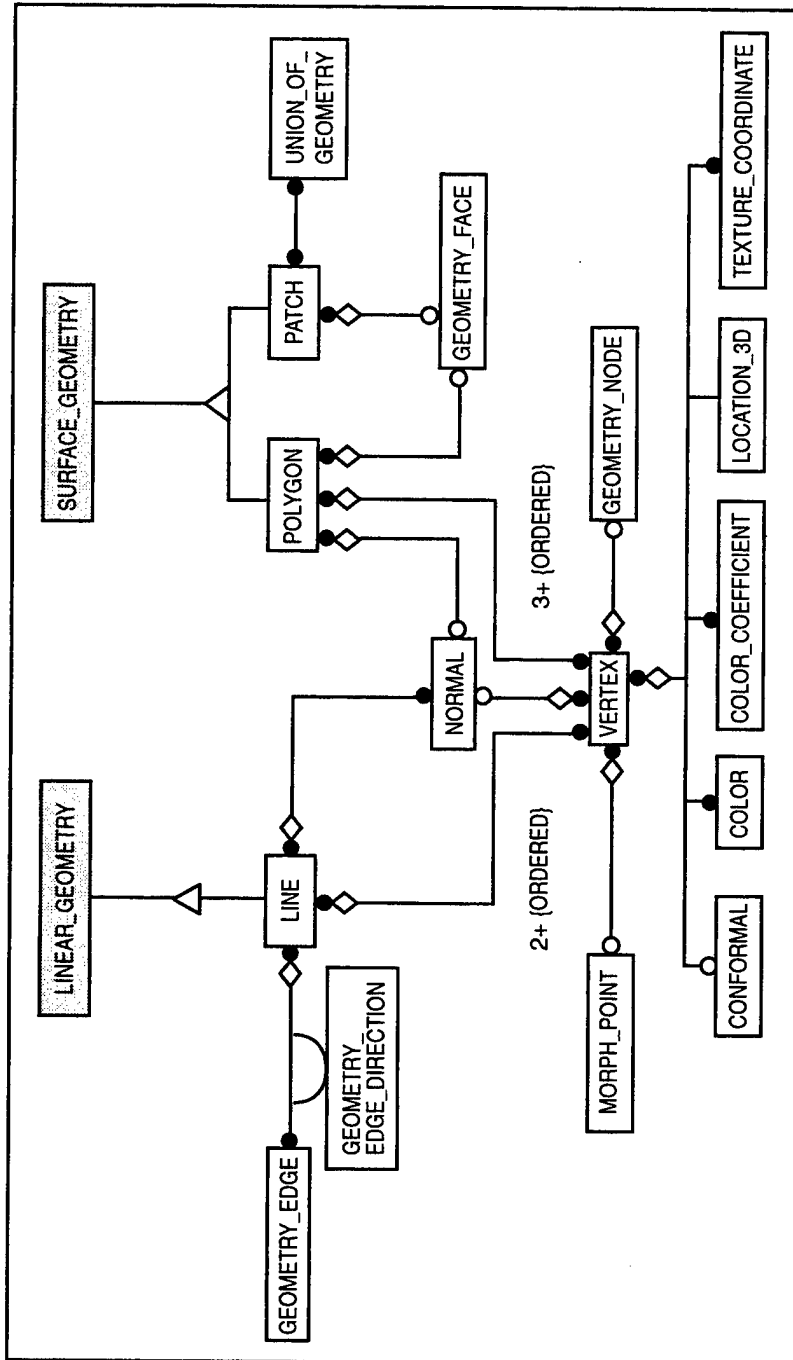
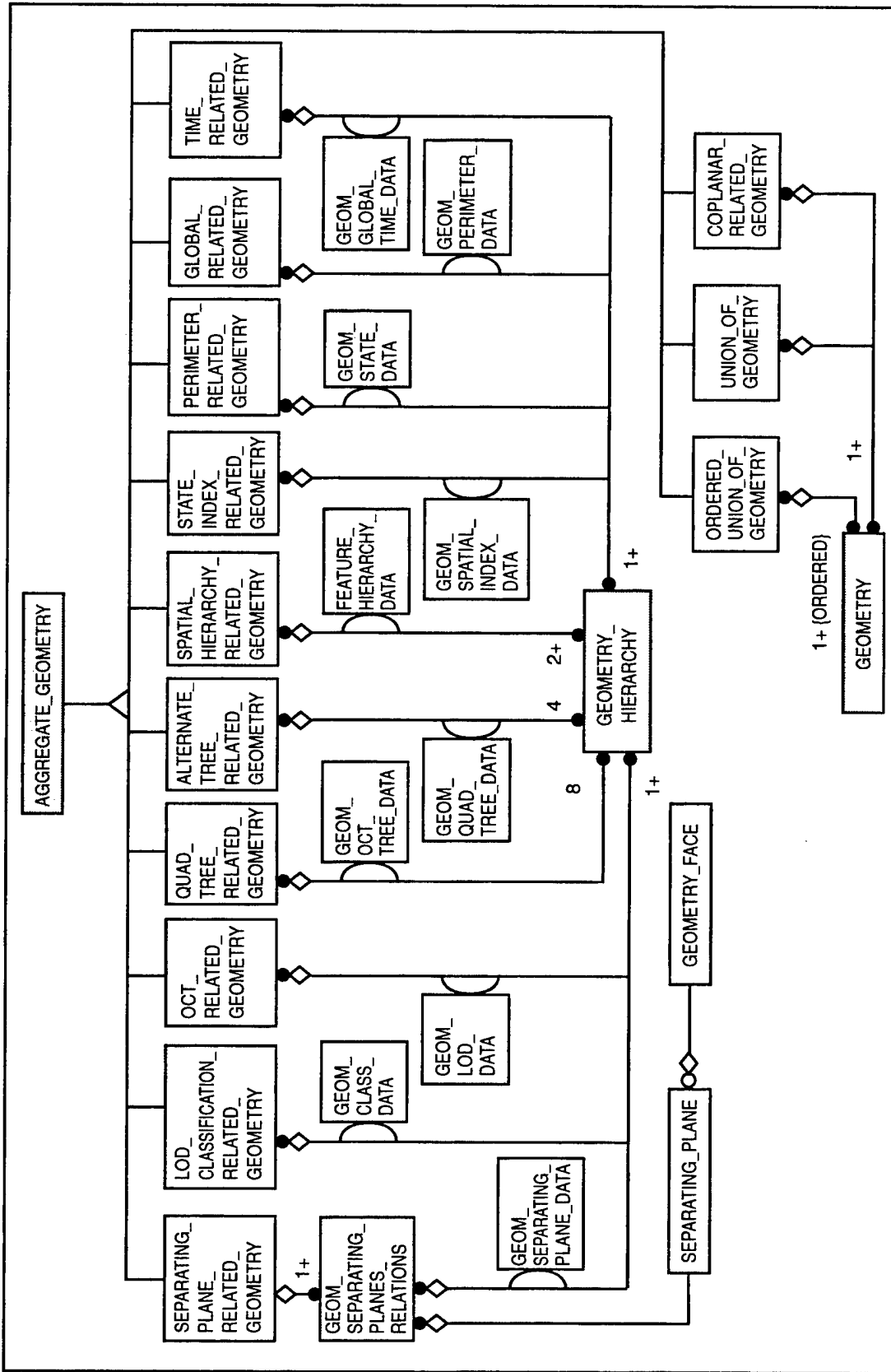Fig. B5 — Linear and surface geometry [1]
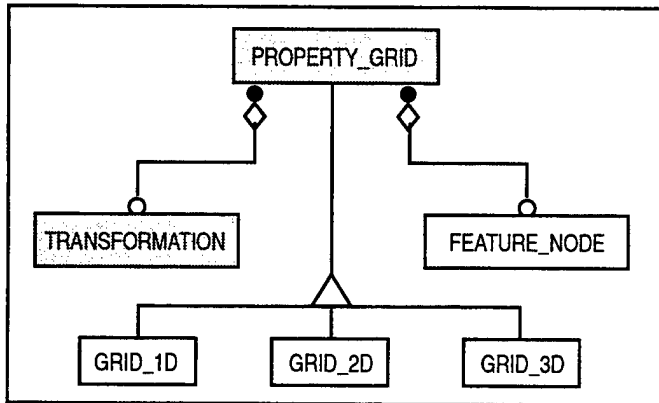
Fig. B6 — Aggregate geometry [1]
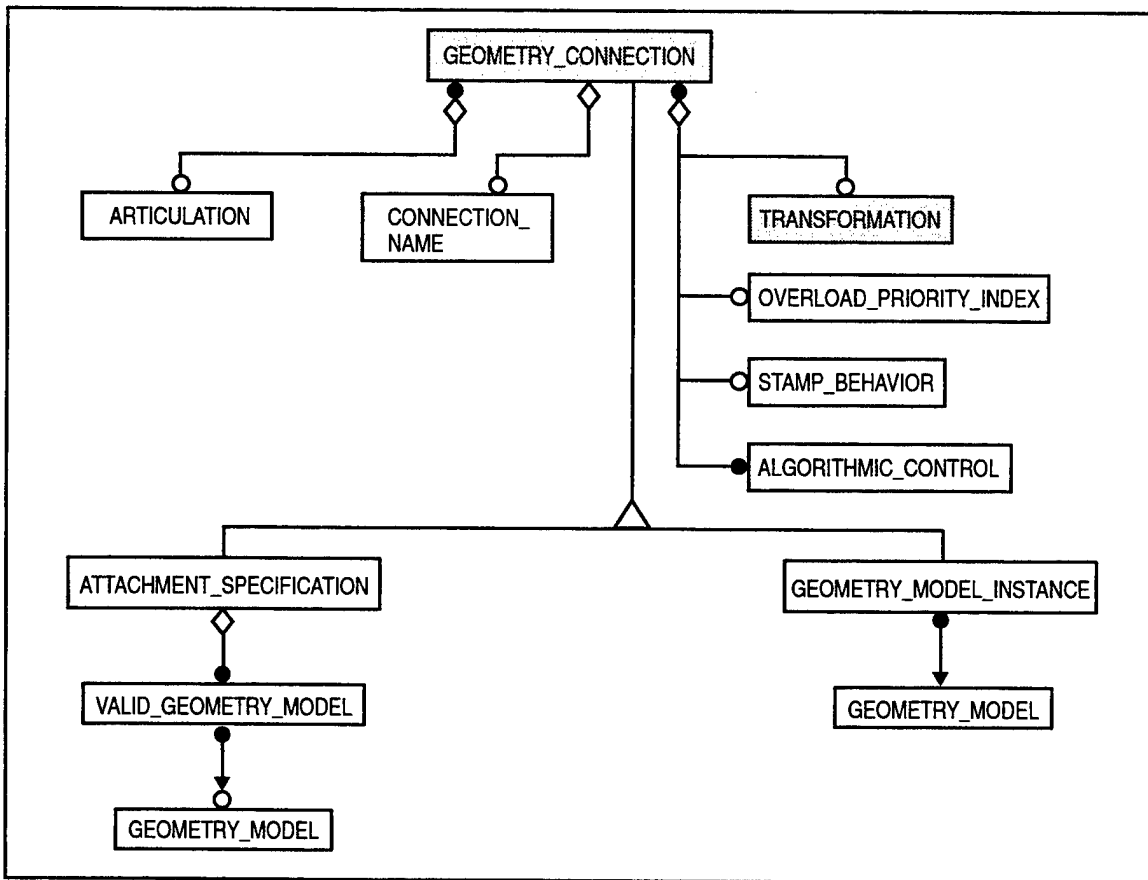
Fig. B7 — Property grids [1]

Fig. B8 — Geometry connection [1]

data necessary to orient objects in the environment. For example, this data might be used to face a building in a certain direction at a certain location.

The final element of Geometry_Hierarchy is Geometry_Connection, shown in Fig. B8. Overload_Priority_Index ranks priority of a model, Stamp_Behavior provides a mechanism for automatically rotating an object to the viewer, and Algorithmic_Control is the function that governs the representation or actions of an object, usually a file name referencing source code.

1.7 Topology

SEDRIS distinguishes Feature_Topology (Fig. B9) and Geometry_Topology (Fig. B10). Both classes are, however, organized around the relationships of nodes, edges, and faces. SEDRIS topology is more fully discussed in Sec. 2.0.

2.0 VPF – SEDRIS DIFFERENCES

This section covers differences between VPF [2] and SEDRIS [1] that are significant to this research. These areas include spatial organization of data (Sec. 2.1), topology (Sec. 2.2), and the class of objects that may be encountered in a SEDRIS transmittal (Sec. 2.3).

2.1 Spatial Organization of Data

VPF, at the coverage level, spatially organizes data into "tiles" that are not subject to further division into smaller tiles. Additionally, all coverages within a library are tiled according to the same tiling scheme. The product specification defines the tiling scheme.

SEDRIS' spatial organization scheme is found under the Aggregate_Geometry and Aggregate_Feature abstract classes through the Spatial_Index_Related_Features and Spatial_Index_Related_Geometry classes (Figs. B3 and B6). Unlike VPF, SEDRIS' spatial index objects are subject to further division into smaller tiles. For example, a Spatial_Index_Related_Feature object can be related to a Feature_Hierarchy, which can have as its children additional Aggregate_Feature objects including further spatial index objects (Fig. B3). This is also true of Spatial_Index_Related_Geometry objects (Fig. B6).

2.2 Topology

Consistent with its division of feature and geometry, SEDRIS distinguishes its treatment of feature topology and geometry topology. Features are required to have topology, but topology is optional for geometric objects.

In VPF as well as in SEDRIS, topology consists of nodes, edges, faces, and their relationships. There are similarities and differences in the way VPF and SEDRIS approach those relationships.

Nodes – In VPF, a node is either a connected node or an entity node. The connected node is related to first edge, and the entity node is related to one containing face. In SEDRIS, a node is not defined to be either "connected" or "entity." Instead, the node can have zero, or at most, one "contained_within" face and zero or more "connected" edges. The contained_within face is then related to one face, and the connected edge is related to one or more ordered edges. The effect is that a node can be an entity node in the sense that it is contained within a face, but also connected to one or more edges.

Edges – VPF's winged-edge topology relates an edge specifically to one left and one right face, to one left and one right edge, and to one start and one end node. SEDRIS topologically relates an edge to one start and to one ending node. The two models topologically diverge beyond that. If
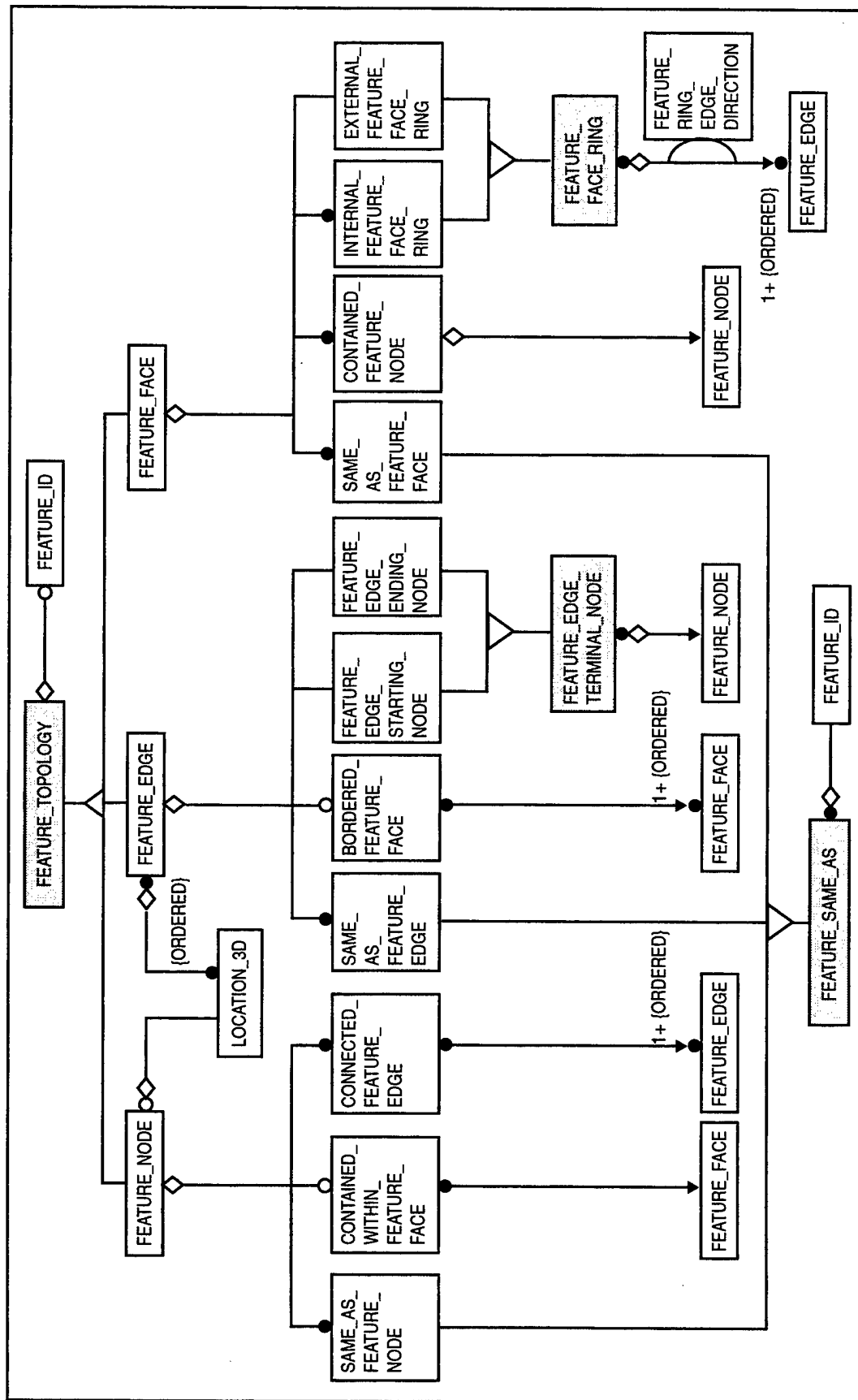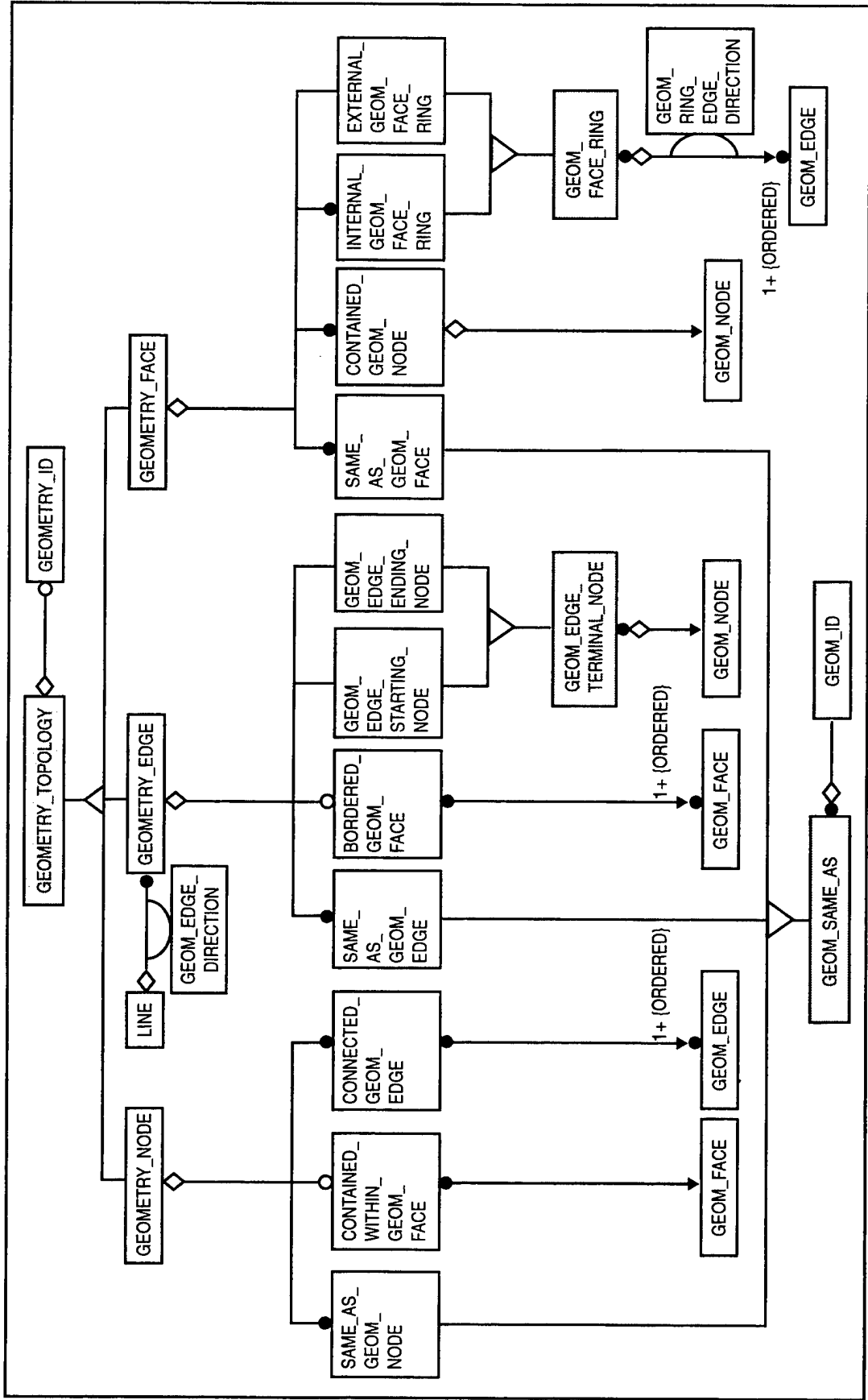
Fig. B9 — Feature topology [1]

Fig. B10 — Geometry topology [1]

an edge has a bordered face in SEDRIS, then that bordered face is related to one or more ordered faces. There is no explicit relationship for a left or a right face. Similarly, SEDRIS provides no explicit relationship for a "left" or a "right" edge.

Faces – VPF relates a face all internal and to one external face ring and then relates each ring to its first edge. SEDRIS similarly relates a face to zero or more internal rings and to one external ring. Each ring is then related to one or more ordered edges. SEDRIS, in addition, relates a face to zero or more contained nodes.

Finally, a node, edge, or face in SEDRIS can be related to a "same as" node, edge, or face. That would allow, for example, the topologies that two identical faces are part of to be joined together by an application.

VPF uses internal and external face rings to topologically organize feature primitives. SEDRIS uses the same tools to topologically organize feature and geometry face primitives. SEDRIS does not provide for a topologic class to describe a 3D volume (such as a "shell" [4]) enclosed by a collection of faces.

2.3 Non-Manifold Objects

Related to topologic considerations of nodes, edges, and faces is the issue of the domain of objects that can be represented by SEDRIS. There is nothing in the SEDRIS data model to prevent the four non-manifold representations shown in Fig. B11.

- Non-manifold edges:
  - Figure B11(a) represents three faces connected along a common edge (more than three are possible);
  - Figure B11(b) shows a "dangling" edge (an edge adjacent to no faces);
  - Figure B11(c) shows an example of a face "dangling" from an open box (three of its edges are adjacent to only one face).
- Non-manifold nodes:
  - Figure B11(b) - the node connecting the dangling edge to the object;
  - Figure B11(d) - the node forming the sole connection between the two objects.
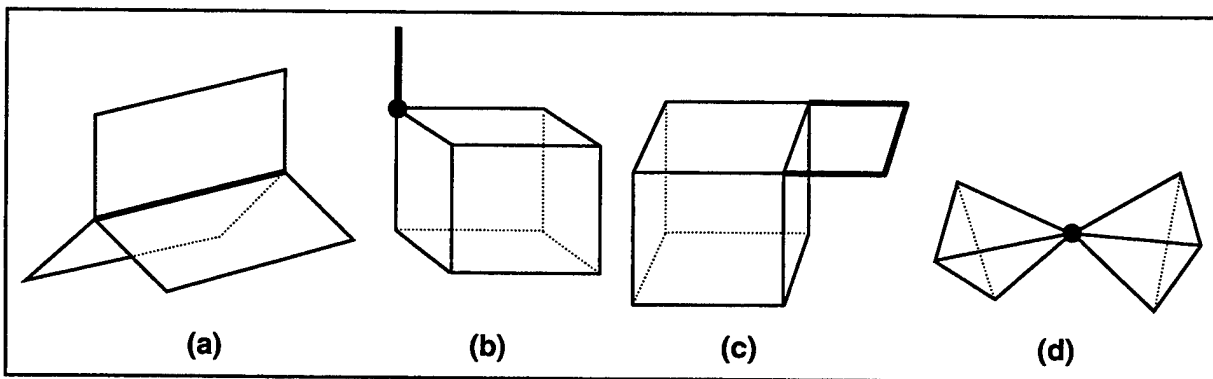

Fig. B11 — Four non-manifold representations

While the topologic organization of SEDRIS would preserve the adjacencies seen in Fig. B11, the edge-based constraints of VPF's winged-edge topology would not. Since VPF's topology specifically relates an edge solely to one left and one right face, a third, fourth, or fifth face would not be retrievable (Fig. B11(a)). Additionally, the edge adjacent to no face (Fig. B11(b)) and the edges adjacent to only one face (Fig. B11(c)) would fail to have the left/right face dichotomy. Finally, VPF only relates a connected node to one "first edge." To find all other edges connected to a node, the topology of that edge is followed until the first edge reappeared. All edges can be located due to face adjacency relationships. If the non-manifold node in Fig. B11(b) were related solely to the dangling edge in the connected node table, then starting from that node, the other three connected edges would not be retrieved. The same would be true given solely the relationship between the non-manifold node and one of the adjoining edges in the box. Similarly, if the non-manifold node in Fig. B11(d) were related solely to one edge in one of the connected objects, then starting from that node, the edges in the other object would not be retrieved to faces.

3.0 REFERENCES

1. SEDRIS Object Model, Release 1.02d.

2. Department of Defense, "Interface Standard for Vector Product Format," MIL-STD-207, 28 Jun 1996.

3. Birkel, P., C. Chiang, S. Farsai, F. Mamaghani, D. Pratt, and J. Smith, "Synthetic Environment Data Representation and Interchange Specification (SEDRIS)," Version: Draft 0.2, 3 Mar 1995.

4. Weiler, K. J., "Topologic Structures for Geometric Modeling," Rensselaer Polytechnic Institute, 1986.
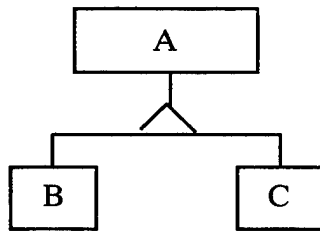
Appendix C

MODIFIED RAUMBAUGH NOTATION

This appendix summarizes the modified Raumbaugh notation used by the SEDRIS object model and is taken from [1].

$$\boxed{\text{A}}$$
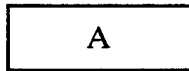
A concrete class showing only its class name.
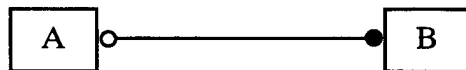Concrete classes are instantiated.

$$\boxed{\text{A}}$$

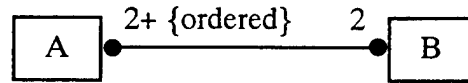Shaded boxes indicate abstract classes that are not instantiated.

Inheritance (is-a) is shown by the triangle.
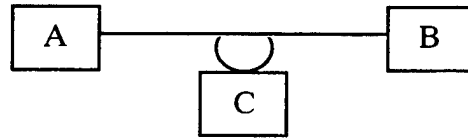Class A is either an object of class B or an object of class C.

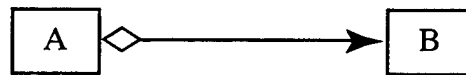Every class A object is related to exactly one class B object.

Every class A object is related to zero or more class B objects.
Every class B object is related to zero or one class A object.

Every class A object is related to two class B objects.
Every class B object is related to two or more ordered class A objects.

Each (A,B) object pair is associated with one class C object.

The diamond shows an aggregation (has-a) relationship.
The arrow indicates a one-way relationship.
An object of class A is composed of one object of class B.
An object of class A knows what object of class B it is associated with.
An object of class B does not know what object of class A it is associated with.

REFERENCES

1. The SEDRIS Team, *Slide Presentation*, spring 1996.