INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 40, 1573–1596 (1997)

A GENERAL TOPOLOGY-BASED MESH DATA STRUCTURE

MARK W. BEALL AND MARK S. SHEPHARD

Scientific Computation Research Center, CII-7011, 110 8th Street, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, U.S.A.

SUMMARY

A representation for a mesh based on the topological hierarchy of vertices, edges, faces and regions, is described. The representation is general and easily supports procedures ranging from mesh generation to adaptive analysis processes. Three implementations are given which concentrate on different aspects of performance (storage requirements and speed). Comparisons are made to other published representations. © 1997 by John Wiley & Sons, Ltd.

KEY WORDS: mesh data structures; topology; hierarchic

1. INTRODUCTION

A critical capability needed by automated, adaptive finite element analysis procedures is the ability to manipulate the mesh of the analysis domain. Data structures have been published for specific parts of the adaptive finite element analysis process, such as searching during mesh generation,[1–3] refinement of existing meshes,[4–9,13–15] and the solution process.[10–12] The present paper focuses on the analysis of a topology-based mesh data structure.

There are weaknesses in the classic element-node mesh data structures, especially in an adaptive analysis environment. One major problem is the lack of information relating the mesh back to the original geometric model. This information, called classification,[16] is critical for mesh generation and enrichment procedures,[17] it allows the specification of analysis attributes in terms of the original geometric model rather than the mesh,[18] and supports direct links to the geometric shape information of the original domain, useful in *p*-version element integration.

An important goal of a mesh data structure is to effectively provide the information required by the procedures that create and/or use that data. The differing needs of these procedures dictate that the database be general and able to answer all queries about the mesh. Such a capability can only be achieved by utilizing a general abstraction of a mesh.[19]

The informational requirements for a general purpose mesh database for automated adaptive finite element analysis on domains defined by manifold geometric models are first given. Then a mesh database based on a topological hierarchy designed to meet these requirements is given. The efficiency of three implementations of this database is discussed with respect to both storage space and access time.

CCC 0029–5981/97/091573–24$17.50

Received 15 October 1995
Revised 28 August 1996

1.1. Nomenclature

Models

- Ω_V domain associated with the model V, $V = G, M$ where G signifies the geometric model and M signifies the mesh model
- $\bar{\Omega}_V$ the closure of the domain associated with the model V, $V = G, M$

Topological entities

- V_i^d the ith entity of dimension d in model V. Shorthand for $V\{V^d\}_i$
- $\partial(V_i^d)$ the entities on the boundary of V_i^d
- $\bar{V}_i^d$ closure of topological entity defined as $V_i^d \cup \partial(V_i^d)$
- $\sqsubset$ classification symbol used to indicate the association of one or more entities from the mesh, M, with an entity in the geometric model, G

Groups

- $\{V^d\}$ unordered group of topological entities of dimension d in model V
- $\lfloor V^d \rfloor$ ordered group of topological entities of dimension d in model V
- $\lceil V^d \rceil$ cyclically ordered group of topological entities of dimension d in model V
- $\langle V^d \rangle$ a group where the ordering is unspecified (ordering is one of: unordered, ordered or cyclically ordered)
- φ_i ith topological entity in group φ, where φ is any one of the groups above

Adjacency operations

- $\varphi\langle V^d \rangle$ the set of entities of dimension d in model V that are adjacent to, or contained in φ. φ may be a single entity, V_i^d or $\langle V^d \rangle_i$, a group of entities, $\langle V^d \rangle$ (possibly a group resulting from another adjacency operation), or a model V
- $\varphi\langle V_{\pm}^d \rangle$ an adjacency relation with directional use information associated with each entity. The $\pm$ indicates the directional use of each entity. A $+$ indicates use in the same direction as the entity definition, a $-$ indicates use in the opposite direction

Examples

- $V\{V^d\}$ all of the entities of order d in model V
- $V_i^{d_i}\{V^{d_j}\}$ the unordered group of topological entities of dimension d_j that are adjacent to the entity $V_i^{d_i}$ in model V
- $V_k^{d_i}\{V^{d_j}\}_i$ the ith member of the unordered group of topological entities of dimension d_j that are adjacent to the entity $V_{k'}^{d_i}$ in model V

The adjacency notation is evaluated from left to right, for example:
$V_i^3\{V^0\}\{V^3\}_j$ is found by first finding $\varphi = V_i^3\{V^0\}$ and then the jth member of $\varphi\{V^3\}$

2. GEOMETRY-BASED AUTOMATED ADAPTIVE ANALYSIS

The goal of an analysis is to solve a set of partial differential equations over a geometric domain, $\bar{\Omega}_G$. Numerical analysis procedures utilize a discretized version of this domain, called a mesh. Since the mesh domain, $\bar{\Omega}_M$, may not be identical to the original geometric domain, $\bar{\Omega}_G$, and various procedures, such as automatic mesh generation, mesh refinement and element stiffness integration, need to understand the relationship of the mesh to the geometric model, it is critical to use a representational scheme which can maintain this relationship. A number of schemes are possible for defining a geometric domain,[20] the most advantageous are boundary-based schemes in which the geometric domain is represented as a set of topological types and adjacencies where

the topological entities have shape information associated with them. Adjacencies are the relationships among topological entities which bound each other. For example, the edges bounding a face is a commonly used topological adjacency.

In addition to being unique, topological entities and their adjacencies provide a convenient abstraction for defining a domain, and allow the convenient specification of analysis attributes such as material properties, loads, boundary conditions and initial conditions with respect to the geometric domain.[18] An additional advantage is that current computer aided design systems support a boundary representation of the domains defined within them, thus allowing efficient combination with automatic mesh generation and modification procedures. Finally, recent boundary representation can properly represent non-manifold geometric domains commonly used for analysis processes.[21,22]

3. REQUIREMENTS FOR MESHES OF MANIFOLD MODELS

This section presents the requirements for manipulating meshes of manifold models. The requirements are fundamentally the same for non-manifold models, however some additions are required for the non-manifold case. For the purposes of clairity these details are not presented here.

3.1. Topological entities

Topology provides an unambiguous, shape independent, abstraction of the mesh. Maintaining the relation between the domain and the mesh is simplified, and many operations can be performed more naturally using the mesh's topological adjacencies.

Each topological entity of dimension d, M_i^d, is bounded by a set of topological entities of dimension $d-1$, $M_i^d\{M^{d-1}\}$. A region is a 3-D entity with a set of faces bounding it. A face is a 2-D entity with a set of edges bounding it. An edge is a 1-D entity with two vertices bounding it.

The representation of general geometric domains requires loop and shell topological entities, and, in the case of non-manifold models, use entities for the vertices, edges, loops, and faces.[21,23] However, restrictions on the topology of a mesh allow a reduced representation in terms of only the basic 0 to d-dimensional topological entities. In three dimensions ($d=3$) these entities are:

$$T_M = \{M\{M^0\}, M\{M^1\}, M\{M^2\}, M\{M^3\}\}$$

where $M\{M^d\}$, $d=0,1,2,3$ are, respectively, the set of vertices, edges, faces and regions defining the primary topological elements of the mesh domain. Restrictions on the topology of a mesh which allow this reduction are:

1. Regions and faces have no interior holes.
2. Each entity of order d_i in a mesh, M^{d_i}, may use a particular entity of lower order, M^{d_j}, $d_j < d_i$, at most once.
3. For any entity $M_i^{d_i}$ there is a unique set of entities of order $d_i - 1$, $M_i^{d_i}\langle M^{d_{i-1}}\rangle$ that are on the boundary of $M_i^{d_i}$ if at least one member of $M_i^{d_i}\langle M^{d_{i-1}}\rangle$ is classifed on $G_j^{d_j}$ where $d_j > d_i$.

The first restriction means that regions may be directly represented by the faces that bound them, and faces may be represented by the edges that bound them. The second restriction allows the orientation of an entity to be defined in terms of its boundary entities (without the

introduction of entity uses). For example, the orientation of an edge, M_i^1 bounded by vertices M_j^0 and M_k^0 is uniquely defined as going from M_j^0 to M_k^0 only if $j \neq k$.

The third restriction means that an interior entity (defined as $M_i^{d_i} \sqsubset G_i^{d_j}$ where, $d_j \geqslant d_i$ and at least one of $\partial(M_i^{d_i}) \sqsubset G_i^{d_j}$ is uniquely specified by its bounding entities. This allows an implementation using a reduced representation for interior entities. This condition only applies to interior entities, entities on the boundary of the model may have a non-unique set of boundary entities as illustrated with a model and a coarse mesh of a plate with a hole in Figure 1. Here, the mesh is sufficiently coarse that the mesh and model topology are identical on the hole boundary. The two mesh edges, M_1^1 and M_2^1, on the hole boundary have the same set of vertices, M_1^0 and M_2^0.

3.2. *Classification*

Classification defines the relationship of the mesh with the geometric domain.

Definition: Mesh Classification Against the Geometric Domain—The unique association of a mesh entity of dimension d_i, $M_i^{d_i}$ to a geometric model entity of dimension d_j, $G_j^{d_j}$ where $d_i \leqslant d_j$, is termed classification and is denoted $M_i^{d_i} \sqsubset G_j^{d_j}$ where the classification symbol, $\sqsubset$, indicates that the left-hand entity, or set, is classified on the right-hand entity.

Multiple $M_i^{d_i}$ can be classified on a $G_j^{d_j}$. Mesh entities are always classified with respect to the lowest-order geometric model entity possible.

Classification of the mesh against the geometric domain is central to (i) ensuring that the automatic mesh generator has created a valid mesh,[16] (ii) transferring analysis attribute information to the mesh,[18] (iii) supporting *h*-type mesh enrichments, and (iv) integrating to the exact geometry as needed by higher-order elements. An example of how classification information is used during the mesh refinement process is illustrated in Figure 2. Figure 2(a) shows the mesh before the dashed edge is split. The model edge is indicated by the bold line. Figure 2(b) shows the mesh after splitting the edge. The classification information is used to recognize that the new

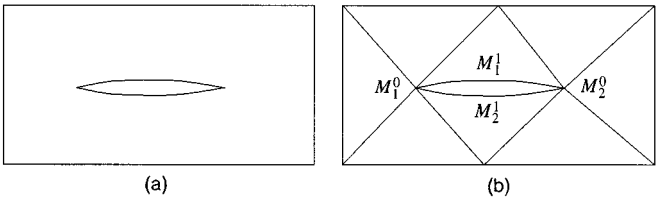

Figure 1. Example of mesh entities on the model boundary having non-unique boundary entities: (a) geometric model; (b) mesh

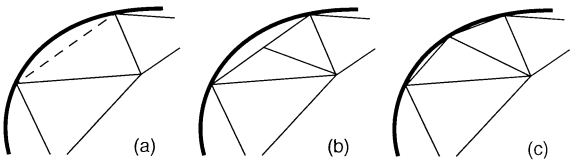

Figure 2. Edge split: (a) before refinement; (b) edge split; (c) new vertex snapped to boundary

vertex created is on the model edge (since the vertex was created by splitting an edge classified on the model edge). The new vertex is then 'snapped' to the boundary so that it is located on the model edge to improve the geometric approximation of the refined mesh.

3.3. Geometric information

The geometric information required for the mesh is limited to pointwise information in terms of the parametric co-ordinates of the model entity that a mesh entity is classified on. Any other shape information can be obtained from the geometric model using the classification information and appropriate queries to the modeler.

3.4. Adjacencies

Adjacencies describe how topological entities connect to each other. There are natural orderings for some adjacencies which prove useful, thus the notation distinguishes between unordered, ordered and cyclic lists. Some adjacencies maintain a directional component that indicates how that entity is used. The right subscript, $\pm$, on the entity, $V^d_{\pm i}$, indicates a directional use of the topological entity as defined by its ordered definition in terms of lower-order entities. A $+$ indicates use in the same direction, while a $-$ indicates use in the opposite direction (e.g. a face, M^2_i, could be defined by the set of edges bounding it as $M^2_i[M^1_{+i}, M^1_{-k}, M^1_{-l}]$ meaning that the edge M^1_j is used in the positive direction, from its first to second vertex, edge M^1_k used in the negative direction and edge M^1_l used in the negative direction).

3.4.1. First-order adjacency relations. The most important set of relations are those which describe, for a given entity $M^{d_i}_k$, all of the entities, $M^{d_j}, (i \neq j)$ which are either on the closure of the entity $(j < i)$, or which it is on the closure of $(j > i)$. These are referred to as first-order adjacencies. For example, the adjacency $M^2_i[M^0]$ is the circular ordered list of all of the mesh vertices which are on the closure of the mesh face M^2_i. The complete list of first-order adjacencies is

Vertex adjacencies:

$$M^0_i\{M^1\}, M^0_i\{M^2\}, M^0_i\{M^3\}$$

Edge adjacencies:

$$M^1_i\lfloor M^0\rfloor, M^1_i\{M^2\}, M^1_i\{M^3\}$$

Face adjacencies:

$$M^2_i[M^0], M^2_i[M^1_\pm], M^2_i\lfloor M^3\rfloor$$

Region adjacencies:

$$M^3_i\{M^0\}, M^3_i\{M^1\}, M^3_i\{M^2_\pm\}$$

Ordered, lower-order adjacencies are used to define the orientation of higher-order entities. The positive orientation of a mesh edge, M^1_i, is defined by the adjacency relation, $M^1_i\lfloor M^0\rfloor$, the positive direction of the edge is from the first vertex, $M^1_i\lfloor M^0\rfloor_0$, to the second vertex, $M^1_i\lfloor M^0\rfloor_1$.

3.4.2. Second-order adjacency relations. Second-order adjacencies describe, for a given entity $M^{d_i}_k$, all of the entities, M^{d_j}, which share any bounding entity of a given order, d_b with the entity. An example of this is the adjacency, $M^3_i\{M^0\}\{M^3\}$, which is the set of all regions which share

a vertex with M_i^3 (useful for element renumbering). The complete set of unordered second-order adjacencies is expressed as follows:

$$M_k^{d_i}\{M^{d_b}\}\{M^{d_j}\},\ d_j \neq d_b,\ d_i \neq d_b$$

As the notation suggests, the second-order adjacencies can be derived from the first-order adjacencies. Higher-order adjacency relations can also be expressed in a similar manner.

3.5. Other Requirements

It must be possible to uniquely associate arbitrary data with each entity to ensure efficiency. For example, traversing the mesh using the mesh adjacencies is made much more efficient by marking entities that have been visited. Other processes can also store data directly on the mesh entities.

Boundary edges and faces must be orientable. Certain mesh generation operations can be made more efficient by ensuring that boundary edges and faces are oriented in the same direction as the model entity that they are classified on.

4. IMPLEMENTATION OPTIONS

The implementation of a mesh database must consider the trade-offs between the storage space required and the time to access various adjacency information. Efficiency dictates that any query be answered by operations whose execution time is not a function of the number of entities in the mesh. Clearly, if more adjacency relations are stored, less work is required to obtain the adjacencies, but the storage space will be greater. The question, then, is which set of adjacencies should be stored for the most efficient implementation.

To avoid global searching to retrieve any adjacency, the graph of the stored adjacencies needs at least one cycle that includes all four of the nodes. Given a set of adjacencies meeting this criteria, how much work must be done to retrieve any adjacency? Stored adjacencies are simply retrieved which is an $O(1)$ operation. Other adjacencies require a local traversal of the graph, these fall into two categories. First, the adjacency desired may be the union of a group of stored adjacencies. For example, if $M_i^3\{M^2\}$ and $M_i^2\langle M^1\rangle$ are stored, then finding $M_i^3\{M^1\}$ requires only collecting the M_i^1 information for each M_i^2 in $M_i^3\{M^2\}$ (finding $M_i^3\{M^2\}\{M^1\}$). The second case is where the union of stored adjacencies is a superset of the entities satisfying the relation, requiring that each entity be examined to determine whether it is to be included. For example, if we have $M_i^3\{M_\pm^2\}$, $M_i^2[M_\pm^1]$, $M_i^1[M^0]$ and $M_i^0\{M^3\}$ and want to find $M_i^2[M^3]$ for some face, $M_i^2[M_\pm^1][M^0]\{M^3\}$ contains regions that do not bound the given face, thus it is necessary to check each region to see if it bounds the face. This is referred to as local searching. Both of these operations are $O(n_e)$, where n_e is the number of entities that must be examined to find the relation. n_e is proportional to the size of the adjacency relation, $n_a, n_e = cn_a$. For the first case, the typical range for c for the relations described in this paper is, $2 < c < 6$. For the second case, the range is $4 < c < 25$ and a check on each entity is required to see if some condition is true.

4.1. Storage requirements

The data structure described here can represent a mesh that is any mixture of various shaped entities (tets, hexes, wedges, pyramids, triangles, quads, lines, etc.). For the purpose of comparing storage requirements, only all tetrahedral and all hexahedral meshes are considered. A mesh that

is a mixture of tetrahedrons, hexahedrons and other common elements would have storage requirements between these two. The number of pointers needed to store each type of connectivity are shown in Table I. This table shows the number of members in the relation $M_k^{d_i}\langle M^{d_j}\rangle$, $(i \neq j)$ for the entire mesh in terms of the number of entities in the mesh. $N_M^d \equiv |M\{M^{d_i}\}|$, is the number of entities of dimension d in model m. For example, in a tetrahedral mesh, there are a total of $4N_M^3$ pointers from regions to faces since each region points to four faces. This means that there are also $4N_M^3$ pointers from faces to regions since each face pointed to by a region points back to that region (although each face only points to two regions).

The relationship between the numbers of the various entities in the mesh is shown in Table II. The tetrahedral mesh is assumed to be infinite with all equilateral tetrahedra (note that this is actually impossible since equilateral tetrahedra do not close pack). These values were checked against real meshes, to check the equilateral assumption gave reasonable results, giving the following ranges: $2{\cdot}02 < N_M^2/N_M^3 < 2{\cdot}19$, $1{\cdot}2 < N_M^1/N_M^3 < 1{\cdot}45$, $0{\cdot}18 < N_M^0/N_M^3 < 0{\cdot}27$, showing reasonably good agreement. For a hexahedral mesh an infinite regular mesh was assumed.

Using the relations in Table II, the adjacency storage requirements (Table I) are rewritten in terms of the number of regions in the mesh (Table III). Table IV shows the average number of adjacencies of each entity type of each other type on a per entity basis. There are many subsets of

Table I. Adjacency storage requirements

	Tetrahedral mesh					Hexahedral mesh			
	M_i^3	M_i^2	M_i^1	M_i^0		M_i^3	M_i^2	M_i^1	M_i^0
M_i^3		$4N_M^3$	$6N_M^3$	$4N_M^3$	M_i^3		$6N_M^3$	$12N_M^3$	$8N_M^3$
M_i^2	$4N_M^3$		$3N_M^2$	$3N_M^2$	M_i^2	$6N_M^3$		$4N_M^2$	$4N_M^2$
M_i^1	$6N_M^3$	$3N_M^2$		$2N_M^1$	M_i^1	$12N_M^3$	$4N_M^2$		$2N_M^1$
M_i^0	$4N_M^3$	$3N_M^2$	$2N_M^1$		M_i^0	$8N_M^3$	$4N_M^2$	$2N_M^1$	

Table II. Relations between number of entities in mesh

Tetrahedral mesh	$N_M^2 \approx 2N_M^3$, $N_M^1 \approx \frac{6}{5}N_M^3$, $N_M^0 \approx \frac{4}{23}N_M^3$
Hexahedral mesh	$N_M^2 \approx 3N_M^3$, $N_M^1 \approx 3N_M^3$, $N_M^0 \approx N_M^3$

Table III. Connectivity storage requirements in terms of regions

	Tetrahedral mesh					Hexahedral mesh			
	M_i^3	M_i^2	M_i^1	M_i^0		M_i^3	M_i^2	M_i^1	M_i^0
M_i^3		$4N_M^3$	$6N_M^3$	$4N_M^3$	M_i^3		$6N_M^3$	$12N_M^3$	$8N_M^3$
M_i^2	$4N_M^3$		$6N_M^3$	$6N_M^3$	M_i^2	$6N_M^3$		$12N_M^3$	$12N_M^3$
M_i^1	$6N_M^3$	$6N_M^3$		$2N_M^3$	M_i^1	$12N_M^3$	$12N_M^3$		$6N_M^3$
M_i^0	$4N_M^3$	$6N_M^3$	$2N_M^3$		M_i^0	$8N_M^3$	$12N_M^3$	$6N_M^3$	

Table IV. Average number of adjacencies per entity $|M_i^{\text{row}}\{M^{\text{col}}\}|$

Tetrahedral mesh					Hexahedral mesh				
	M^3	M^2	M^1	M^0		M^3	M^2	M^1	M^0
M_i^3		4	6	4	M_i^3		6	12	8
M_i^2	2		3	3	M_i^2	2		4	4
M_i^1	5	5		2	M_i^1	4	4		2
M_i^0	23	35	14		M_i^0	8	12	6	

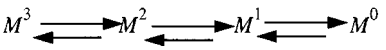

Figure 3. Graph of stored adjacencies for one-level adjacency representation

the first-order adjacencies from which the remaining adjacencies can be derived. The next three sections give implementations that match well with the various requirements for retrieving information used in automated adaptive analysis processes.

4.2. *One-level adjacency representation*

One possible adjacency set is to maintain adjacencies between entities one dimension apart. A data structure similar to this is discussed in Reference 24 for the specific case of tetrahedral meshes. Figure 3 graphically depicts this set of relationships.
The actual adjacencies stored are

Downward adjacencies:

$$M_i^1\lfloor M^0\rfloor, M_i^2[M_\pm^1], M_i^3\{M_\pm^2\}$$

Upward adjacencies:

$$M_i^0\{M^1\}, M_i^1\{M^2\}, M_i^2\lfloor M^3\rfloor$$

The missing relations can be reconstructed as

$$M_i^3\{M^1\} = M_i^3\{M^2\}\{M^1\}, \quad M_i^3\{M^0\} = M_i^3\{M^2\}\{M^1\}\{M^0\}$$

$$M_i^0\{M^3\} = M_i^0\{M^1\}\{M^2\}\{M^3\}, \quad M_i^0\{M^2\} = M_i^0\{M^1\}\{M^2\}$$

$$M_i^1\{M^3\} = M_i^1\{M^2\}\{M^3\}$$

$$M_i^2[M^0] = \{M_i^2[M_{\alpha_n}^1]_n\lfloor M^0\rfloor_{\beta_n}\}, \quad \beta_n = \begin{cases} 0, & \alpha_n = + \\ 1, & \alpha_n = - \end{cases}, \quad n = 0 \ldots |M_i^2[M^1]|$$

The last expression deserves an explanation. Each edge in the adjacency $M_i^2[M_\pm^1]$ is examined in order. One vertex from each edge is added to the set based on the direction the face is using the edge. If edge $M_i^2[M_\pm^1]_n$ is used in the $+$ direction, the first vertex is taken, if it is used in the $-$ direction, the second vertex is taken. This results in the ordered set of vertices around the face.

The time to retrieve the unstored relations is less than implied by the operators above. For example, obtaining $M_i^3\{M^0\} = M_i^3\{M^2\}\{M^1\}\{M^0\}$ for a tetrahedron requires looking at only two of the faces of the region. The first face yields three of the vertices of the region and any other face gives the fourth. Similar processes can be determined for some of the other relations. Table V shows an estimate of the operation count to obtain each of the adjacency relations for the one-level representation.

4.3. *Circular adjacency representation*

Another reasonable set of adjacencies is to store downward pointers from each entity to the entity one dimension lower and to store pointers from the vertices up to the highest-order entities that are using them (in a 3-D manifold mesh this would be the mesh regions) as shown in Figure 4. Less information is stored in this scheme, however more work must be done to obtain the upward adjacencies that are not being stored. The actual adjacencies stored are

Downward adjacencies:

$$M_i^1[M^0], M_i^2\langle M_{\pm}^1\rangle, M_i^3\{M_{\pm}^2\}$$

Upward adjacencies:

$$M_i^0\{M^3\}$$

This set of relations has the minimum connectivity storage in which all entities are explicitly represented. This can be seen by weighting the corresponding edges in the graph of first-order adjacencies with the connectivity storage requirements in Table III. This set of relations (or the similar one using the inverse of each relation: $M_i^0\{M^1\}$, $M_i^1\{M^2\}$, $M_i^2\lfloor M^3\rfloor$ and $M_i^3\{M^0\}$) is the minimum weighted cyclic path that includes all four nodes in the graph. The three downward and one upward adjacencies are used instead of three upward and one downward since there is directional use information in the downward relations.

Finding the missing relations is more involved than with the one level adjacency relations. Procedures for constructing the relations $M_i^0\{M^1\}$, $M_i^1\{M^2\}$, and $M_i^2\lfloor M^3\rfloor$ are shown

Table V. Operation count for retrieving adjacency $M_i^{\text{row}}\{M^{\text{col}}\}$ for one-level representation

	Tetrahedral mesh					Hexahedral mesh			
	M^3	M^2	M^1	M^0		M^3	M^2	M^1	M^0
M_i^3		1	9	6	M_i^3		1	20	16
M_i^2	1		1	3	M_i^2	1		1	4
M_i^1	10	1		1	M_i^1	8	1		1
M_i^0	140	70	1		M_i^0	48	24	1	

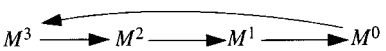

Figure 4. Graph of adjacencies for circular adjacency representation

below. The remaining adjacency relations are found as shown previously for the one-level representation.

$$M_i^0\{M^1\}\colon M_j^1 \in M_i^0\{M^1\} \quad \text{if } M_j^0 \in \partial(M_j^1) \quad \text{where } M_j^1 = M_i^0\{M^3\}\{M^2\}\{M^1\}_j$$

$$M_i^1\{M^2\}\colon M_j^2 \in M_i^1\{M^2\} \quad \text{if } M_j^1 \in \partial(M_j) \quad \text{where } M_j = M_i^1\{M^0\}\{M^3\}\{M^2\}_j$$

$$M_i^2\lfloor M^3 \rfloor\colon R = M_i^2\{M^1\}\{M^0\}\{M^3\} \ (R \text{ is the set of all regions bounding the closure of } M_i^2)$$

Each region in R must be checked to determine if it is adjacent to M_i^2. The direction that the region is using M_i^2 determines which side of M_i^2 the region is on:

$$M_i^2\lfloor M^3 \rfloor_0 = R_j \text{ such that } R_j\{M_{\beta_k}^2\}_k = M_i^2 \text{ and } \beta_k = -$$

$$M_i^2\lfloor M^3 \rfloor_1 = R_j \text{ such that } R_j\{M_{\beta_k}^2\}_k = M_i^2 \text{ and } \beta_k = +$$

Table VI shows an estimate of the operation count needed to retrieve each adjacency. Most of the upward adjacencies require a local search consisting of traversing the entire cycle in the adjacency graph then doing topological queries on each entity that is found. Thus, the time required to determine these relations is larger than for the one-level adjacency set.

4.4. Reduced representations

The restriction: For any entity $M_i^{d_i}$ there is a unique set of entities of order $d_i - 1$, $M_i^{d_i}\langle M^{d_{i-1}}\rangle$ that are on the boundary of $M_i^{d_i}$ if at least one member of $M_i^{d_i}\langle M^{d_{i-1}}\rangle$ is classified on $G_i^{d_j}$ where $d_j > d_i$, requires interior entities to be uniquely defined by their boundary entities. This allows the elimination of interior faces and edges without losing any information about the mesh.

The functionality presented earlier must not be affected by the elimination of entities. Although the implementation does not explicitly represent these entities, the interface must act as though it does. All the operations given earlier must be possible even for entities which are not explicitly represented. The general idea is that if an entity that is not represented and the program using the database needs that entity (e.g. the entity is returned as a part of an adjacency relation) a temporary proxy is returned for the entity. The lifetime of this proxy is only as long as the program is referencing that entity. There are two important issues in eliminating entities: reconstruction of the eliminated entities as needed and associating data with the eliminated entities.

Table VI. Operation count for retrieving adjacency $M_i^{\text{row}}\{M^{\text{col}}\}$ for circular representation

	Tetrahedral mesh					Hexahedral mesh			
	M^3	M^2	M^1	M^0		M^3	M^2	M^1	M^0
M_i^3		1	9	6	M_i^3		1	20	16
M_i^2	299		1	3	M_i^2	148		1	4
M_i^1	538	570		1	M_i^1	304	296		1
M_i^0	1	264	304		M_i^0	1	192	228	

4.4.1. Reconstructing eliminated entities. The most complex aspect of reconstructing eliminated entities is the fact that edges and faces are oriented entities which must always have a consistent orientation. That is, if an edge, M_i^1, which is not explicitly stored is defined as $M_i^1\lfloor M_j^0, M_k^0\rfloor$ at one point it must never be redefined as $M_i^1\lfloor M_k^0, M_j^0\rfloor$ on a later query. The same consistency of ordering of edges around a face applies.

One way in which this can be accomplished is (Fig. 5):

1. Assign each vertex in the mesh a unique number (integer).
2. Define edge M_i^1 as going from M_j^0 to M_k^0 where $j < k$. That is, edges which are not explicitly represented are defined such that the positive direction of the edge is from the lower numbered vertex to the higher numbered one.
3. A face M_i^2 is defined in terms of an ordered set of vertices $M_i^2[M^0]$ where the face orientation is defined by the loop in the direction from the lowest numbered vertex to the next lowest numbered vertex adjacent to it.

Although a unique orientation for edges and faces is obtained, the ability to arbitrarily orient them is lost. Since the orientation of an interior edge or face is determined by the numbering of the vertices (which is hidden from the programmer) an edge defined from vertex M_k^0 to vertex M_j^0, $M_i^1\lfloor M_k^0, M_j^0\rfloor$, may actually end up being oriented as $M_i^1\lfloor M_j^0, M_k^0\rfloor$ if $j < k$. This cannot be changed by simply renumbering the vertices as demonstrated in Figure 6. However, since there is no requirement to orient internal edges and faces this is a workable approach.

It is necessary to know which vertices are used to define the edges and faces. This information comes from the region definition. In order to infer the existence of faces and edges the relation $M_i^3\lfloor M^0\rfloor$ (an ordered set corresponding to $M_i^3\{M^0\}$) must be defined for each type of region (e.g. hexahedron, tetrahedron, wedge, etc.). This requirement was not present in the previous representations where the topological configuration of the region did not need to be explicitly stored.

4.4.2. Associating data with eliminated entities. It is not possible to store data on eliminated entities. The best way to resolve this is to store the data associated with the edges and faces on the

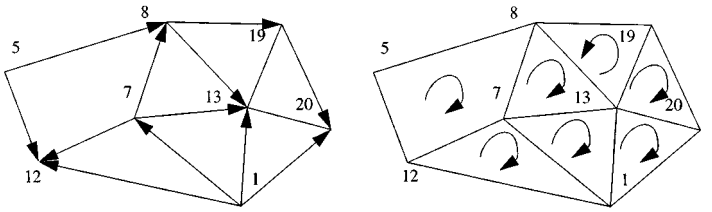

Figure 5. Edge and face orientations based on vertex numbering

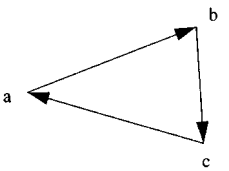

Figure 6. Impossible edge orientation, requires $a < b$, $b < c$, $c < a$

vertices used to define them. To do this not only the data must be stored but also information that indicates which face or edge the data belongs to (for an edge it would be necessary to store the other vertex, for a face all the other vertices which define the face). This extra information used to indicate the owner of the data is only needed when there is data actually stored on the entity.

4.4.3. Implementation. Elimination of faces and edges only in the interior of the mesh gives a data structure called the reduced interior representation. On the boundary $M_i^{d_i} \sqsubset G_i^{d_i}$ is represented. The adjacency graph of this representation is shown in Figure 7.

The adjacency graph is more complicated since the mesh representation is now heterogeneous. The dashed lines in the graph indicate adjacencies that are implicitly stored due to the ordering of vertices defining a region. The ordering of vertices which defines the faces must not be used for faces on the boundary. Local searching must be done to find these faces (which are represented). It can be seen from the adjacency graph that any downward adjacency can be directly retrieved. Upward adjacencies are obtained in a similar manner to the circular hierarchic representation. Table VII shows an estimate of the operation count to retrieve adjacencies for the reduced representation. The counts shown only consider retrieving adjacencies for interior entities, more searching must be done on the boundary to find boundary entities. It is assumed that it takes one operation to construct a proxy for an entity that is not explicitly represented. The operation counts are less than those for the circular representation since all of the downward adjacencies are stored (either explicitly or implicitly). The retrieval operations that require local searching take more time than the same operation using the one-level adjacency representation.

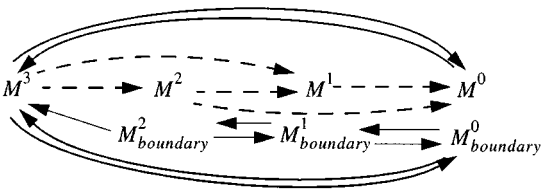

Figure 7. Adjacency graph for reduced interior representation

Table VII. Operation count for retrieving adjacency $M_i^{\text{row}}\{M^{\text{col}}\}$ for reduced representation

	Tetrahedral mesh					Hexahedral mesh			
	M^3	M^2	M^1	M^0		M^3	M^2	M^1	M^0
M_i^3		4	6	1	M_i^3		8	12	1
M_i^2	293		3	1	M_i^2	176		4	1
M_i^1	230	373		1	M_i^1	112	212		1
M_i^0	1	219	198		M_i^0	1	116	86	

4.4.4. Issues with the reduced representation. There are some issues with the reduced representation that make them less desirable than the representations with a complete set of entities.[19] The most important problem is that modifying the mesh may change the definition of mesh entities that were not directly modified. For example, Figure 8 shows an edge collapse procedure where this redefinition occurs. The dashed edge is collapsed with vertex 6 replacing vertex 2. The two figures show the edge orientations (as arrows on the edges) and the face orientations (a + face has its normal pointing out of the page, a − face has it pointing into the page) as given by the vertex numbering scheme. After the edge collapse, three things have happened that would not happen if the edges and faces were directly represented: (i) two edges (2–4 and 2–3) have actually changed their identities causing any references to them to become invalid, (ii) those same two edges have changed their orientations (thus the direction that the faces are using them have changed), and (iii) a face (previously 1–2–4, now 1–4–6) has changed its orientation. The same operation using a representation with all entities present would have resulted in only adjacency information being changed. The entities themselves would have remained unchanged including their orientations.

This non-local effect makes this representation less efficient for doing many mesh modifications since information about the topology of the mesh that is saved by the procedure may become invalid when an operation on the mesh is performed. This means that the procedure must reacquire this information after each mesh modification. With the full representation of all entities, the propagation of these changes is very limited and predictable.

5. COMPARISON TO CLASSIC FE DATA STRUCTURE

This section compares the size of a data structure based on the classic element-node connectivity to the hierarchic representations showing that the hierarchic data structure does not necessarily take significantly more storage space than a classic data structure, especially when other data structures needed to perform an analysis are considered. The comparison is not really fair since the classic data structure does not meet the needs of various adaptive procedures. Meshes consisting of tetrahedral and hexahedral elements of up to cubic order are considered. For the purposes of the comparison, only serendipity elements with nodes on edges are considered, although all the data structures can easily store any type of element. For this comparison sizes are given in words, where an integer or a pointer is one word and a real value is 2 words.

5.1. Classic mesh data structure

The classic approach to a mesh data structure describes the mesh in terms of elements and nodes. Additional data structures needed for operations such as node or element reordering are

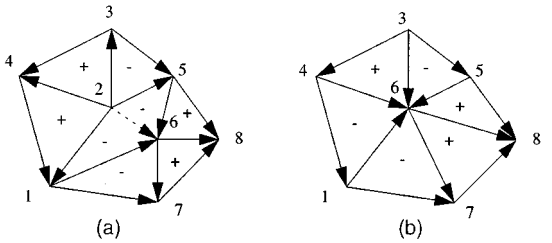

Figure 8. Edge collapse: (a) before collapse; (b) after collapse

also commonly constructed when this data structure is used. An element is defined by an ordered list of nodes (Figure 9). Each node has an id and a position in space.

The size of the Element data structure is $n + 2$ where n is the number of nodes in the element. The size of the Node data structure is 7. Table VIII shows the number of nodes (N) as a function of the number of edges and vertices in a mesh.

Given the size and number of nodes and elements, the storage needed for the mesh can be calculated as shown in Table IX.

5.1.1. Equation renumbering. Some of the most successful renumbering algorithms are Sloan, Gibbs–King, Gibbs–Poole–Stockmeyer (GPS) and reverse Cuthill–McKee (see Reference 25 and references therein). All of these algorithms build a graph of the node-to-node connectivity of the mesh. An efficient implementation uses an adjacency list accessed by a pointer vector[26] requiring storage of $2E + N$ words, where E is the number of edges and N is the number of nodes in the graph. Other storage is also needed which varies greatly by algorithm. The

```
Element {
    int type;
    ptr attributes;
    ptr nodes[n]
}
n = 4(linear tet.) 10 (quad. tet.), 16 (cubic tet.), 8 (linear hex), 20
(quad. hex.), 32 (cubic hex.)
```

```
Node {
    int id;
    real x,y,z;
}
```

Figure 9. Classic mesh data structure

Table VIII. Number of nodes and elements in mesh

Element type	Number of nodes	Number of elements
Linear	${}_mN^0$	N_M^3
Quadratic	${}_mN^1 + {}_mN^0$	N_M^3
Cubic	$2{}_mN^1 + {}_mN^0$	N_M^3

Table IX. Total storage by entity—classic

	Element	Node	Total
		Tetrahedral	
Linear	$6N_M^3$	$1N_M^3$	$7N_M^3$
Quadratic	$12N_M^3$	$8N_M^3$	$20N_M^3$
Cubic	$18N_M^3$	$15N_M^3$	$33N_M^3$
		Hexahedral	
Linear	$10N_M^3$	$7N_M^3$	$17N_M^3$
Quadratic	$22N_M^3$	$28N_M^3$	$50N_M^3$
Cubic	$34N_M^3$	$49N_M^3$	$83N_M^3$

number of nodes, N, in the adjacency graph is the same as the number of finite element nodes in the mesh. The number of edges in the graph depends on the type of mesh and on the particular mesh itself.

E can be calculated for various types of meshes. For each node, the number of connected graph edges, E_n, is the number of nodes on all the elements that share that node (counting each node only once). For linear hexahedral elements (nodes only at the vertices) the connectivity of each node is all the nodes on the eight hexahedral elements that meet at each vertex (26). E_n for various types and orders of meshes is shown in Table X.

The value of E is the connectivity of each node times the number of nodes of that type (Table XI). Note the dramatic increase in connectivity information that must be stored for higher order elements. These numbers indicate that in these situations it may be wise to avoid these renumbering schemes which are derived solely from the structure of the assembled system of equations and use an approach based on the connectivity of the mesh as described in Section 6.5.

5.2. *Hierarchic data structure—one-level*

A data structure for the one-level representation is shown in Figure 10. The number of upward pointers from edges to faces and from vertices to edges is the average number of entities in that adjacency relation. The size of each entity can be calculated as shown in Table XII. Total storage for the mesh is shown broken down by mesh entity in Table XIII.

5.3. *Hierarchic data structure—circular*

The hierarchic data structure for the circular representation is shown in Figure 11. A real implementation would have to be slightly more complicated to handle mesh generation and adaption procedures where a partially constructed mesh may exist.

Table X. Node connectivity

	Tetrahedral		Hexahedral	
Element order	Vertex nodes	Edge nodes	Vertex nodes	Edge nodes
Linear	14	N/A	26	N/A
Quadratic	61	22	80	50
Cubic	107	38	130	81

Table XI. Total connectivity storage

Element type	Tetrahedral	Hexahedral
Linear	$2{\cdot}5N_M^3$	$26N_M^3$
Quadratic	$37N_M^3$	$230N_M^3$
Cubic	$64N_M^3$	$371N_M^3$


```
Region {
    ptr classification;
    int #faces;
    ptr faces[4^t or 6^h];
}

Face {
    ptr classification;
    int #edges;
    ptr edges[3^t or 4^h];
    ptr regions[2];
}

Edge {
    ptr classification;
    ptr vertices[2];
    int #faces;
    ptr faces[5^t or 4^h]
    int node_id[0^l,1^q or 2^c];
    Point node_location[0^l,1^q or 2^c];
}

Vertex {
    ptr classification;
    #edges;
    edges[14^t or 6^h]
    int node_id;
    Point location;
}

Point {
    real x,y,z;
}

Meaning of superscripts:
t: tetrahedral mesh
h: hexahedral mesh
l: linear mesh
q: quadratic mesh
c: cubic mesh
```

Figure 10. Hierarchic data structure—one-level

Table XII. Entity sizes for one-level representation

	Tetrahedral				Hexahedral			
	Region	Face	Edge	Vertex	Region	Face	Edge	Vertex
Linear	6	7	9	23	8	8	8	15
Quadratic	6	7	16	23	8	8	15	15
Cubic	6	7	23	23	8	8	22	15

Table XIII. Total storage by entity—one-level representation

	Region	Face	Edge	Vertex	Total
			Tetrahedral		
Linear	$6N_M^3$	$14N_M^3$	$11N_M^3$	$4N_M^3$	$35N_M^3$
Quadratic	$6N_M^3$	$14N_M^3$	$19N_M^3$	$4N_M^3$	$43N_M^3$
Cubic	$6N_M^3$	$14N_M^3$	$28N_M^3$	$4N_M^3$	$52N_M^3$
			Hexahedral		
Linear	$8N_M^3$	$24N_M^3$	$24N_M^3$	$15N_M^3$	$71N_M^3$
Quadratic	$8N_M^3$	$24N_M^3$	$45N_M^3$	$15N_M^3$	$92N_M^3$
Cubic	$8N_M^3$	$24N_M^3$	$66N_M^3$	$15N_M^3$	$113N_M^3$

The sizes of each entity are shown in Table XIV. In comparison to the one-level representation the region is the same size, the face and edge structures are smaller (since they do not have upward connectivity stored) and the vertex is larger (since the number of regions adjacent to a vertex is larger than the number of edges adjacent to a vertex). The overall storage broken down by entity is shown in Table XV. The total storage is 15–25 per cent less than the one-level representation.


```
Region {
    ptr classification;
    int #faces;
    ptr faces[4^t or 6^h];
}

Face {
    ptr classification;
    int #edges;
    ptr edges[3^t or 4^h];
}

Edge {
    ptr classification;
    ptr vertices[2];
    int #faces;
    int node_id[0^l,1^q or 2^c];
    Point node_location[0^l,1^q or 2^c];
}

Vertex {
    ptr classification;
    #regions;
    regions[23^t or 8^h]
    int node_id;
    Point location;
}

Point {
    real x,y,z;
}

Meaning of superscripts:
t: tetrahedral mesh
h: hexahedral mesh
l: linear mesh
q: quadratic mesh
c: cubic mesh
```

Figure 11. Hierarchic data structure—circular

Table XIV. Hierarchic representation entity sizes—circular

	Tetrahedral				Hexahedral			
	Region	Face	Edge	Vertex	Region	Face	Edge	Vertex
Linear	6	5	4	32	8	6	4	17
Quadratic	6	5	11	32	8	6	11	17
Cubic	6	5	18	32	8	6	18	17

Table XV. Total storage by entity—circular

	Region	Face	Edge	Vertex	Total
			Tetrahedral		
Linear	$6N_M^3$	$10N_M^3$	$5N_M^3$	$5N_M^3$	$26N_M^3$
Quadratic	$6N_M^3$	$10N_M^3$	$13N_M^3$	$5N_M^3$	$34N_M^3$
Cubic	$6N_M^3$	$10N_M^3$	$22N_M^3$	$5N_M^3$	$43N_M^3$
			Hexahedral		
Linear	$8N_M^3$	$18N_M^3$	$12N_M^3$	$17N_M^3$	$55N_M^3$
Quadratic	$8N_M^3$	$18N_M^3$	$33N_M^3$	$17N_M^3$	$76N_M^3$
Cubic	$8N_M^3$	$18N_M^3$	$48N_M^3$	$17N_M^3$	$91N_M^3$

5.4. Hierarchic data structure—reduced interior representation

The data structure for the reduced interior representation (Figure 12) is more complicated than the other two hierarchic representations. There are two different representations of the vertex, one for the boundary and one for the interior. The vertices stored in the region must be stored in a known order for each element topological configuration (e.g. tetrahedron, hexahedron). The data structure shown assumes that there is a full reprsentation on the boundary (edges classified


```
Region {
    ptr classification;
    int type;
    ptr vertices[4^t or 8^h];
}

Boundary Face {
    ptr classification;
    int #edges;
    ptr edges[3^t or 4^h];
    ptr regions[2];
}

Boundary Edge {
    ptr classification;
    ptr vertices[2];
    int #faces;
    ptr faces[2]
    int node_id[0^l,1^q or 2^c];
    Point node_loc[0^l,1^q or 2^c];
}

Boundary Vertex {
    ptr classification;
    # b_edges;
    ptr b_edges[6^t or 4^h]
    int node_id;
    Point location;
    int #regions;
    ptr regions[12^t or 4^h];
    int #interior edges;
    Edge_info edges[4^t or 1^h];
}

Vertex {
    ptr classification;
    #regions;
    regions[23^t or 8^h]
    int node_id;
    Point location;
    Edge_info edges[7^t or 3^h]
}

Edge_info{
    ptr other_vertex;
    int node_id[1^q or 2^c]
    Point[1^q or 2^c];
}

Point {
    real x,y,z;
}

Meaning of superscripts:
t: tetrahedral mesh
h: hexahedral mesh
l: linear mesh
q: quadratic mesh
c: cubic mesh
```

Figure 12. Hierarchic data structure—reduced interior

Table XVI. Entity sizes—reduced interior

	Region	Boundary face	Boundary edge	Boundary vertex	Vertex
			Tetrahedral		
Linear	6	7	6	29	32
Quadratic	6	7	13	61	88
Cubic	6	7	20	89	137
			Hexahedral		
Linear	10	8	6	19	17
Quadratic	10	8	13	27	41
Cubic	10	8	20	34	62

on model faces are represented). The implementation shown here is a little simpler than would be needed for mesh generation since it would be necessary to be able to represent a partially constructed mesh.

The sizes of each entity are given in Table XVI. Compared to the other two hierarchic representations most of the data has been moved to the vertex. Part of the reason for this is that information that was stored on the edges (nodes in this case) is now stored on one of the vertices of the edge.

To calculate the total storage for this representation (Table XVII) the percentage of boundary entities must be known. For the comparison used here it is assumed that the mesh has 30 per cent of its vertices, 10 per cent of its edges and 5 per cent of its faces on the boundary.[19]

5.5. *Node renumbering with the hierarchic mesh representations*

Extra data structures for node (or element) renumbering are not needed with the hierarchic mesh representation since the needed adjacency information is already available. One simple procedure for nodal renumbering uses the adjacency information to traverse the mesh, numbering the nodes as it does so. A small amount of storage is needed for the queue and to find a good starting set of vertices, but extra storage for the node-to-node connectivity is not needed. Experience has shown that this type of renumbering results in global stiffness matrices with bandwidths competitive with those generated by other renumbering algorithms. In fact, the algorithm is much the same as reverse Cuthill–McKee,[25] it is even possible to add degree of node priority to this algorithm making it even more like reverse Cuthill–McKee.

```
initialize queue with vertices
current_node_number = number of nodes
while queue not empty{
    remove first vertex from queue
    number node at vertex with current_node_number
    current_node_number = current_node_number − 1
    for each unnumbered node on any higher order entities adjacent to vertex{
        number node with current_node_number
        current_node_number = current_node_number − 1
    }
    add neighboring vertices of vertex that are not in queue to queue
}
```

This type of renumbering could also be used with a classic data structure. It would require building the node-element connectivity for the vertex nodes only.

Table XVII. Total storage by entity—reduced interior

	Region	Boundary Face	Boundary Edge	Boundary Vertex	Vertex	Total
			Tetrahedral			
Linear	$6N_M^3$	$0{\cdot}5N_M^3$	$1N_M^3$	$1{\cdot}5N_M^3$	$4N_M^3$	$13N_M^3$
Quadratic	$6N_M^3$	$0{\cdot}5N_M^3$	$1{\cdot}5N_M^3$	$3N_M^3$	$11N_M^3$	$22N_M^3$
Cubic	$6N_M^3$	$0{\cdot}5N_M^3$	$2{\cdot}5N_M^3$	$5N_M^3$	$17N_M^3$	$31N_M^3$
			Hexahedral			
Linear	$10N_M^3$	N_M^3	$2N_M^3$	$6N_M^3$	$12N_M^3$	$31N_M^3$
Quadratic	$10N_M^3$	N_M^3	$4N_M^3$	$8N_M^3$	$29N_M^3$	$52N_M^3$
Cubic	$10N_M^3$	N_M^3	$6N_M^3$	$10N_M^3$	$44N_M^3$	$71N_M^3$

5.6. Size comparison

The information from the previous sections is summarized in Table XVIII and Table XIX. The cost of the hierarchic data structures decreases rapidly as higher-order elements are used. If renumbering is taken into account the hierarchic data structures are smaller for quadratic and higher-order elements.

Another interesting result can be found by normalizing the mesh sizes by the number of nodes in the mesh (Table XX). Since the number of nodes in the mesh is related to the amount of information stored on the mesh during the solution process, this can be viewed as the information cost of the mesh. Doing this normalization allows meshes of different element orders and element types to be compared. Increasing the element order decreases the information cost since the fixed cost of storing the mesh topology is amortized over more nodes. One interesting observation is the high information cost of a linear tetrahedral mesh compared to a linear hexahedral mesh and that the large difference virtually disappears when the order of each mesh is raised to quadratic.

Another item that must be considered is that, during a solution process, other information must be stored in addition to the mesh. At a minimum, a certain number of degrees of freedom per node are stored. At most, all the local stiffness matrices may be stored. Somewhere in the middle, in terms of storage, would be storing the assembled stiffness matrix in some form. This storage is relevant since, if it is large compared to the mesh storage required, a small amount of extra storage for the mesh is not very significant.

Table XVIII. Size comparison—tetrahedral meshes (numbers in parenthesis are classic data structure with renumbering information)

Element order	Classic	One-level	% of classic	Circular	% of classic	Reduced interior	% of classic
Linear	$7N_M^3$ ($9{\cdot}5N_M^3$)	$35N_M^3$	500% (368%)	$26N_M^3$	371% (274%)	$13N_M^3$	186% (137%)
Quadratic	$29N_M^3$ ($57N_M^3$)	$43N_M^3$	215% (75%)	$34N_M^3$	170% (60%)	$22N_M^3$	110% (39%)
Cubic	$33N_M^3$ ($97N_M^3$)	$52N_M^3$	158% (54%)	$43N_M^3$	130% (44%)	$31N_M^3$	94% (32%)

Table XIX. Size comparison—hexahedral meshes (numbers in parenthesis are classic data structure with renumbering information)

Element order	Classic	One-level	% of classic	Circular	% of classic	Reduced interior	% of classic
Linear	$17N_M^3$ ($43N_M^3$)	$71N_M^3$	418% (165%)	$55N_M^3$	324% (128%)	$31N_M^3$	182% (72%)
Quadratic	$50N_M^3$ ($280N_M^3$)	$92N_M^3$	184% (33%)	$76N_M^3$	152% (27%)	$52N_M^3$	104% (19%)
Cubic	$83N_M^3$ ($454N_M^3$)	$113N_M^3$	136% (25%)	$91N_M^3$	110% (20%)	$71N_M^3$	86% (16%)

Table XX. Information cost (words/node) (numbers in parenthesis are classic data structure with renumbering information)

Element order	Classic	One-level	Circular	Reduced interior
		Tetrahedral mesh		
Linear	40 (56)	201	153	76
Quadratic	15 (41)	31	25	16
Cubic	13 (37)	20	17	12
		Hexahedral mesh		
Linear	17 (43)	71	55	31
Quadratic	13 (70)	23	19	13
Cubic	12 (65)	16	13	10

Table XXI. Information cost for solution data structures (words/node) n is the number of degrees of freedom per node

Element order	Solution	Element matrices	Global stiffness
		Tetrahedral mesh	
Linear	$2n$	$376n^2$	$21n^2$
Quadratic	$2n$	$292n^2$	$41n^2$
Cubic	$2n$	$398n^2$	$64n^2$
		Hexahedral mesh	
Linear	$2n$	$256n^2$	$39n^2$
Quadratic	$2n$	$400n^2$	$86n^2$
Cubic	$2n$	$585n^2$	$132n^2$

A summary of this storage is given in Table XXI for three different cases. First, the storage for the degrees of freedom that hold the solution itself is fixed at 2 words per degree of freedom (one double precision number). Second, storage for the individual element matrices is given. Third, the storage needed for an assembled global stiffness matrix using compressed row storage[27] is given assuming a symmetric system. The compressed row storage is likely to be the most compact storage possible for the global matrix. In particular, a skyline storage requires storage per node equal to the average bandwidth of the matrix which will increase as the mesh is refined for a given problem, the compressed row storage per node is independent of the problem size. The last two items in Table XXI depend on the square of the number of degrees of freedom per node, since the size of the stiffness matrix of an element scales in this manner. Note that while the information cost for the mesh decreases as the element order is increased the information cost for the solution generally increases.

A comparison to the mesh storage necessitates picking specific problem types and solution procedures. For illustrative purposes a 3-D elasticity problem (three degrees of freedom per node) using quadratic tetrahedral elements and a solver that uses an assembled global stiffness matrix

(compressed row storage), is considered. The total storage for the solution process will be 375 words/node (6 words/node for the solution and 369 words/node for the global stiffness matrix). The classic data structure adds another 15 words/node for a total of 390 words/node. The largest hierarchic data structure adds 31 words/node for a total of 406 words/node. This 4 per cent increase in storage (1 per cent for hexahedral elements) for using the richer data structure for the mesh is not significant.

6. COMPARISON TO SPECIAL PURPOSE HIERARCHIC DATA STRUCTURES

There have been some published hierarchic data structures used in adaptive analysis that, although not entirely general purpose, are well suited to the functions required by the specific procedures. This section investigates the storage penalty incurred by using the general purpose data structure described here versus one specifically designed for the problem. All of the data structures presented were specifically designed to handle only tetrahedral meshes which saves some storage space since the number of downward adjacencies is fixed.

6.1. Edge-based data structure

Biswas and Strawn[14] present a data structure tailored to an edge-based analysis and refinement scheme. This data structure is a cross between the one-level adjacency structure and the reduced interior representation. Their data structure omits interior faces but includes faces classified on the boundary (Figure 13). Interior and boundary edges are included. Their data structure does not have classification information.

A calculation of the size of their data structure, including only the mesh information (not the solution storage which is also given in their paper) gives a storage of $22{\cdot}5N_M^3$. This is between the circular and reduced-interior representations.

6.2. Data structure with fast retrieval of downward adjacencies

Kallinderis and Vijayan present a data structure containing all four topological mesh entities and primarily downward adjacency information (Figure 14) in Reference 13. Retrieving some adjacencies with this data structure would require global searching, however their adaptive analysis does not need these adjacencies.

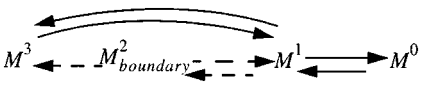

Figure 13. Data structure of Biswas and Strawn[14]

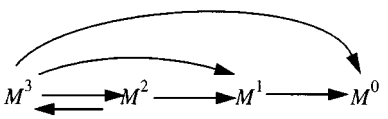

Figure 14. Data structure of Kallinderis and Vijayan[13]

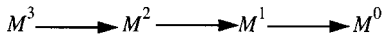

Figure 15. Data structure of Connell and Holmes[15]

This structure is optimized for speed since it explicitly stores the adjacencies required by their adaptive procedure, rather than deriving them from other adjacencies. Their data structure also does not have classification information. The size of their data structure is $27N_M^3$ which is roughly the same as the circular representation of the hierarchic data structure.

6.3. Data structure with only downward adjacencies

Connell and Holmes give a data structure in Reference 15 that provides only downward adjacencies (Figure 15). They do however include classification information and correctly reposition vertices classified on model boundaries during mesh refinement. Again, such a data structure would require global searching for some adjacencies which are apparently not needed by their analysis. The storage for this representation requires $17{\cdot}5N_M^3$. This is half of the one-level adjacency structure and between the storage for the circular and reduced-interior representations.

7. CLOSING REMARKS

Requirements for a general purpose mesh database based on a hierarchy of topological entities were presented. Three implementations that meet these requirements were given and compared to the classic element-node representation typically used in finite element analysis codes. It was shown that when all the storage needed for the solution process was considered, the hierarchic representation does not add a significant amount of extra storage.

The hierarchic representations add important information and capabilities that are needed for other parts of an adaptive analysis environment. These representations maintain the relation of the mesh to the geometric model that it was created from which is critical for mesh generation and enrichment procedures. This representation can easily be extended to properly represent meshes of non-manifold models.

REFERENCES

1. R. Löhner, 'Some useful data structures for the generation of unstructured grids', *Comm. appl. numer. methods*, **4,** 123–135 (1988).
2. J. Bonet and J. Peraire, 'An alternating digital tree (ADT) algorithm for 3D geometric and intersection problems', *Int. j. numer. methods eng.*, **31,** 1–17 (1991).
3. H. Dannelongue and P. Tanguy, 'Efficient data structure for adaptive remeshing with the FEM', *J. Comput. Phys.*, **91,** 94–109 (1990).
4. G. F. Carey, M. Sharma and K. C. Wang, 'A class of data structures for 2-D and 3-D adaptive mesh refinement', *Int. j. numer. methods eng.*, **26,** 2607–2622 (1988).
5. M. C. Rivara, 'Design and data structure of fully adaptive, multigrid, finite element software', *ACM Trans. Math. Soft.*, **10,** 242–264 (1984).
6. W. C. Rheinboldt and C. K. Mesztenyi, 'On a data structure for adaptive finite element mesh refinements', *ACM Trans. Math. Soft.*, **6,** 166–187 (1980).
7. P. Devloo, J. T. Oden and T. Strouboulis, 'Implementation of an adaptive refinement technique for the SUPG algorithm', *Comput. Methods Appl. Mech. Eng.*, **61,** 339–358 (1987).
8. N. Golias and T. Tsiboukis, 'An approach to refining three-dimensional tetrahedral meshes based on Delaunay transformations', *Int. j. numer. methods eng.*, **37,** 793–812 (1994).
9. K. C. Chellamuthu and N. Ida, 'Algorithms and data structures for 2D and 3D adaptive finite element mesh refinement', *Finite Elements Anal. Des.*, **17,** 205–229 (1994).
10. R. Löhner, 'Edges, starts, superedges and chains', *Comput. Methods Appl. Mech. Eng.*, **111,** 255–263 (1994).

11. R. Löhner, 'Some useful renumbering strategies for unstructured grids', *Int. j. numer. methods eng.*, **36,** 3259–3270 (1993).
12. D. M. Hawken, P. Townsend and M. F. Webster, 'The use of dynamic data structures in finite element applications', *Int. j. numer. methods eng.*, **33,** 1795–1811 (1992).
13. Y. Kallinderis and P. Vijayan, 'Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes', *AIAA J.*, **31,** 1440–1447 (1993).
14. R. Biswas and R. Strawn, 'A new procedure for dynamic adaption of three-dimensional unstructured grids', *AIAA-93-0672*, Presented at the 31st Aerospace Sciences Meeting & Exhibit, Jan. 11–14, 1993, Reno, NV.
15. S. D. Connell and D. G. Holmes, '3-dimensional unstructured adaptive multigrid scheme for the euler equations', *AIAA J.*, **32,** 1626–1632 (1994).
16. W. J. Schroeder and M. S. Shephard, 'A combined octree/delaunay method for full automatic 3-D mesh generation', *Int. j. numer. methods eng.*, **29,** 37–55 (1990).
17. M. S. Shephard and M. K. Georges, 'Reliability of automatic 3D mesh generation', *Comput. Methods Appl. Mech. Eng.*, **101,** 443–462 (1992).
18. M. S. Shephard, 'The specification of physical attribute information for engineering analysis', *Eng. Comput.*, **4,** 145–155 (1988).
19. M. Beall and M. Shephard, 'Mesh data structures for advanced finite element applications', *SCOREC Report 23-1995*, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy NY, 1995.
20. A. A. G. Requicha and H. B. Voelcker, 'Solid modeling: current status and research directions', *IEEE Comput. Graphics Appl.*, **3,** 25–37 (1983).
21. K. J. Weiler, 'The radial-edge structure: a topological representation for non-manifold geometric boundary representations', in M. J. Wozny, H. W. McLaughlin and J. L. Encarnacao (eds.), *Geometric Modeling for CAD Applications*, North-Holland, Amsterdam, 1988, pp. 3–36.
22. E. L. Gursoz, Y. Choi and F. B. Prinz, 'Vertex-based representation of non-manifold boundaries', in M. J. Wozny, J. U. Turner and K. Priess (eds.), *Geometric Modeling Product Engineering*, North-Holland, Amsterdam, 1990, pp. 107–130.
23. K. J. Weiler, 'Topological structures for geometric modeling', *Ph.D. Thesis*, Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy NY, May 1986.
24. E. Bruzzone, L. De Floriani and E. Puppo, 'Manipulating three-Dimensional triangulations', in Lecture Notes in Computer Science, Vol. 367, Springer, Berlin, 1989, pp. 339–353.
25. L. T. Souza and D. W. Murray, 'A unified set of resequencing algorithms', *Int. j. numer. methods eng.*, **38,** 565–581 (1995).
26. S. W. Sloan and W. S. Ng, 'A direct comparison of three algorithms for reducing profile and wavefront', *Comput. Struct.*, **33,** 411–419 (1989).
27. I. Duff, R. Grimes and J. Lewis, 'Sparse matrix test problems', *ACM Trans. Math. Soft.*, **15,** 1–14 (1989).

