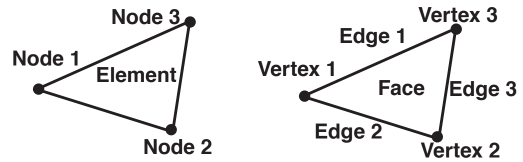
Hierarchical Mesh Representation

- Element-Node connectivity is not sufficient for mesh generation and adaption - Can't be used to verify mesh validity
- Operations on other entities (such as faces or edges) are often more efficient and natural
- Topological hierarchy gives a general, shape-independent abstraction of a mesh



- Also a useful representation for analysis procedures
- Reference: M.W. Beall and M.S. Shephard, "A General Topology-Based Mesh Data Structure," *Int. J. Num. Meth. Engng.*, 40(9):1573-1596, 1997.

Topological Entities

Topology provides an unambiguous, shape independent, abstraction of the mesh

Each topological entity of dimension d, M_i^d, is defined by a set of topological entities of dimension $d-1$, $M_i^d\{M^{d-1}\}$, which form its boundary.

A region is a 3-d entity defined by the set of faces that bound it.

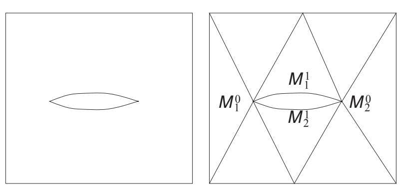
A face is a 2-d entity defined by the set of edges that bound it.

An edge is a 1-d entity defined by the two vertices that bound it.

A vertex is a 0-d entity that is the base of the hierarchy, it has no lower order entities bounding it.

Mesh Topology

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 on the model entity $G_j^{d_j}$ where $d_j \geq d_i$.



(a) Geometric Model (b) Mesh

Non-Manifold Models

Supporting the representation of meshes of non-manifold models gives the ability to represent:

Discontinuous Fields

- Contact
- Material Interfaces

Representational Incompatibilities in FE model

- Interface between different idealizations (shell/solid)
- Solid/Fluid interfaces

Both cases have different d.o.f. on each side of interface

Models with mixed dimension entities

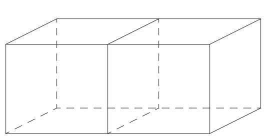
- Solid (3D), shell (2D) and wire (1D) entities

All give need to differentiate between different uses of a face or edge

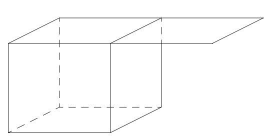
Non-Manifold Geometric Models

Non-manifold models are common in engineering analysis

Two common situations that result in non-manifold models are:



Material Interfaces



Dimensional Reductions

A hierarchic representation of the mesh allows these to be dealt with in a natural manner

Mesh Classification

Definition: Mesh Classification Against the Geometric Domain - The unique association of a topological mesh entity to a topological geometric domain entity.

$M_i^{d_i} \sqsubset G_i^{d_i}$ denotes $M_i^{d_i}$ is classified on $G_i^{d_i}$, $(d_i \leq d_j)$

- Multiple $M_i^{d_i}$ can be classified on a $G_i^{d_i}$.

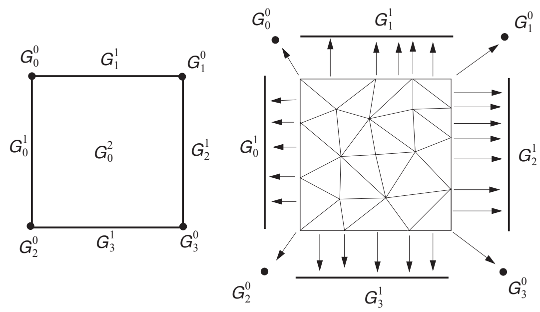
  A mesh region, M_i^3, is classified on a G_i^3

  A mesh face, M_i^2, is classified on a G_i^3, or G_i^2.

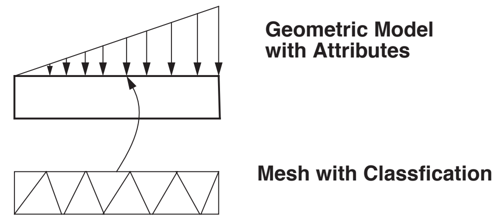
  A mesh edge, M_i^1, is classified on a G_i^3, G_i^2, or G_i^1

  A mesh vertex, M_i^0, is classified on a G_i^3, G_i^2, G_i^1, or G_i^0
- Mesh entities are always classified with respect to the lowest order object entity possible.

Mesh Classification



Classification and Attributes



- Attributes which apply to mesh are retrieved using classification information
- Attributes are never “transferred” to the mesh - always are only defined on geometric model
- When mesh is adapted, attributes do not change

Radial Edge Data Structure

MODEL

Region
↕
Shell
↕
Face use ↔ Face
↕
Loop use ↔ Loop
↕
Edge use ↔ Edge
↕
Vertex use ↔ Vertex

MESH

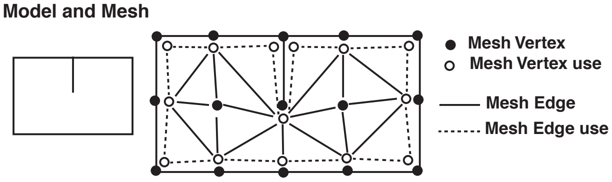
Region
↕
Face use ↔ Face
↕
Edge use ↔ Edge
↕
Vertex use ↔ Vertex

Minimal Use Mesh Data Structure

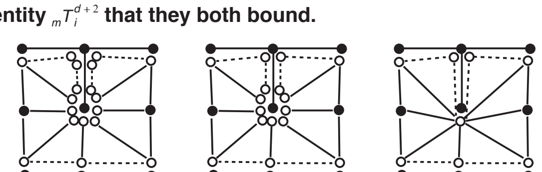
- Full non-manifold representation not necessary for meshes
  Contains redundant information since classification against the non-manifold model is available for use information
  Takes too much space (many use entities)
- Can eliminate many uses - Minimal Use Mesh Data Structure
  Maintains use entities only on mesh entities classified on the boundary of the model
  Multiple use entities used in the mesh only when necessary to distinguish between separate use entities in the geometric model
  Elimination of redundant uses can be done by considering mesh topology and classification information

Plate with crack

Model and Mesh



- Two uses of entity can be condensed to one use if there is an entity ${}_mT_i^{d+2}$ that they both bound.



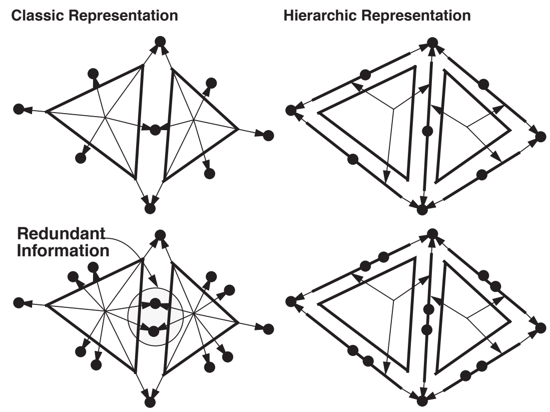
Requirements for Meshes of Manifold Models

1. It must be possible to iterate through all the entities of a given type in a mesh.
2. It must be possible to compare two entities to see if they are the same.
3. It must be possible to retrieve the classification of any mesh entity.
4. It must be possible to store spatial locations in terms of parametric coordinates on each mesh entity.
5. All adjacencies must be retrievable for any mesh entity.
6. It must be possible to uniquely associate arbitrary data with each entity.
7. Boundary edges and faces must be orientable.

Application to Analysis Codes

- The hierarchical mesh representation can also be used for analysis
- Rather than having elements and nodes, degrees of freedom are directly associated with mesh entities
- Allows same representation to be used for mesh modification and for analysis - important in adaptive environments
- Reduces redundant information storage in higher order formulations
  - Multiple nodes on mesh edge or face are pointed to by each element in classic representation
  - Hierarchic representation only points to each once
  - Very important for variable order p-meshes
- Provides links to exact geometry for element integration procedures

Redundant information: Higher-orders nodes on shared edge



First-Order Adjacencies

First-order adjacencies for $M_k^{d_i}$ are the entities, M^{d_j}, $(i \neq j)$ which are either on its closure $(j < i)$ or which it is on the closure of $(j > i)$.

The complete list of these adjacencies is as follows:

Vertex adjacencies: $M_i^0\{M^1\}$, $M_i^0\{M^2\}$, $M_i^0\{M^3\}$

Edge adjacencies: $M_i^1\lfloor M^0 \rfloor$, $M_i^1[M^2]$, $M_i^1[M^3]$

Face adjacencies: $M_i^2[M^0]$, $M_i^2[M^1_\pm]$, $M_i^2\lfloor M^3 \rfloor$

Region adjacencies: $M_i^3\{M^0\}$, $M_i^3\{M^1\}$, $M_i^3\{M^2_\pm\}$

$\lfloor\ \rfloor$ - ordered list, $[\ ]$ - ordered cyclic list, $\{\ \}$ - unordered list

Storing all relations would take up too much space.

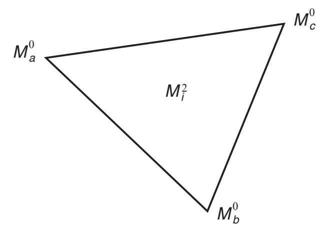
Can derive some of the above relations from the others.

e.g. can derive $M_i^3\{M^1\}$ from $M_i^3\{M^2_\pm\}$ and $M_i^2[M^1_\pm]$

Three reasonable implementations will be given that satisfy all requirements

First-Order Adjacencies

Example: $M_i^2[M^0]$ is the circular ordered list of mesh vertices which are on the closure of the mesh face M_i^2.



$$M_i^2[M^0] = [M_a^0, M_b^0, M_c^0]$$

Second-Order Adjacencies

Second-order adjacencies of $M_k^{d_i}$ are all of the entities, M^{d_j}, which share a boundary entity of a given order, d_b with the entity.

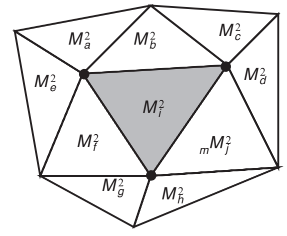
The complete set of unordered second-order adjacencies can be expressed as follows:

$$M_i^j\{M^k\}\{M^l\}, j \neq k, l \neq k$$

- Second order adjacencies are derivable from first order adjacencies.
- Higher order adjacency relations can be expressed in a similar manner.

Second-Order Adjacencies

Example: $M_i^2\{M^0\}\{M^2\}$, which is the set of all faces which share a vertex with M_i^2 (such a relationship is useful for element renumbering).



$$M_i^2\{M^0\}\{M^2\} = \{M_a^2, M_b^2, M_c^2, M_d^2, M_e^2, M_f^2, M_g^2, M_h^2, M_j^2\}$$

Implementation Options

One-Level Representation

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

- all relations are easy/fast to obtain
- not minimum storage

$$M^3 \rightleftarrows M^2 \rightleftarrows M^1 \rightleftarrows M^0$$

Circular Representation

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

- adjacency relations can be derived
- less storage than one-level representation
- upward adjacency relations are more costly to obtain than from one-level adjacency

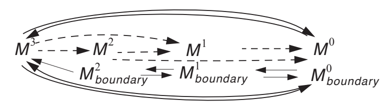
$$M^3 \rightarrow M^2 \rightarrow M^1 \rightarrow M^0 \rightarrow M^3$$

Reduced Interior Representation

Only $M_i^3\lfloor M^0\rfloor$, $M_i^0\{M^3\}$ **on interior**

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

- interior faces and edges not explicitly represented
- ordered region-vertex relation implies interior entities
- orientation of interior entities determined by a vertex numbering scheme
- less storage than either circular or one-level representations
- inefficient for procedures that modify mesh



Performance Comparison

Operation count to retrieve adjacency relation - tetrahedral mesh

	One-level	Circular	Reduced interior	Classic
$M_i^0\{M^1\}$	1	304	198	n.a.
$M_i^0\{M^2\}$	70	264	219	n.a.
$M_i^0\{M^3\}$	140	1	1	n.a.
$M_i^1\lfloor M^0\rfloor$	1	1	1	1
$M_i^1[M^2]$	1	570	373	n.a.
$M_i^1[M^3]$	10	538	230	n.a.
$M_i^2[M^0]$	3	3	1	1
$M_i^2[M_\pm^1]$	1	1	3	3
$M_i^2\lfloor M^3\rfloor$	1	299	293	n.a.
$M_i^3\{M^0\}$	6	6	1	1
$M_i^3\{M^1\}$	9	9	6	6
$M_i^3\{M_\pm^2\}$	1	1	4	4

n.a. - cannot be obtained without global search

Performance Comparison

Operation count to retrieve adjacency relation - hexahedral mesh

	One-level	Circular	Reduced interior	Classic
$M_i^0\{M^1\}$	1	228	86	n.a.
$M_i^0\{M^2\}$	24	192	116	n.a.
$M_i^0\{M^3\}$	48	1	1	n.a.
$M_i^1\lfloor M^0\rfloor$	1	1	1	1
$M_i^1[M^2]$	1	296	212	n.a.
$M_i^1[M^3]$	8	304	112	n.a.
$M_i^2[M^0]$	4	4	1	1
$M_i^2[M_\pm^1]$	1	1	4	4
$M_i^2\lfloor M^3\rfloor$	1	148	176	n.a.
$M_i^3\{M^0\}$	16	16	1	1
$M_i^3\{M^1\}$	20	20	12	12
$M_i^3\{M_\pm^2\}$	1	1	8	8

n.a. - cannot be obtained without global search

Size Comparison

Comparison to published adaptive data structures shows hierarchic representation is approximately the same size (and is more general)

For analysis purposes comparison to classic mesh data structure is of interest

Not fair comparison since, classic mesh data structure:

- is not suited to needs of adaptivity
  - no classification information
  - insufficient representation of mesh to verify that mesh correctly represents the geometric model
- is not well suited for variable p-meshes
- needs auxiliary data structures for operations such as node renumbering - hidden cost that can be huge

Classic Mesh Data Structure

- Stores Element-Node connectivity

```
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;
}
```

- Node reordering storage based on storage needed to build up node-to-node connectivity graph

  typical implementation of Sloan, Gibbs-King, Gibbs-Poole-Stockmeyer and reverse Cuthill-McKee procedures

One-Level Hierarchic 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];
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 superscipts:
t: tetrahedral mesh
h: hexahedral mesh
l: linear mesh
q: quadratic mesh
c: cubic mesh

Circular Hierarchic 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 superscipts:
t: tetrahedral mesh
h: hexahedral mesh
l: linear mesh
q: quadratic mesh
c: cubic mesh

Reduced Interior Representation:

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]
}

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

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

Point {
real x,y,z;
}

Size Comparison

Tetrahedral Meshes

Element Order	Classic	One-Level	% of Classic	Circular	% of Classic	Reduced Interior	% of Classic
Linear	$7N_M^3$ ($9.5N_M^3$)	$35N_M^3$	500% (368%)	$26N_M^3$	371% (274%)	$13N_M^3$	186% (137%)
Quadratic	$20N_M^3$ ($57N_M^3$)	$43N_M^3$	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%)

(parenthesis indicate size with data structures for nodal renumbering)

Size Comparison

Hexahedral Meshes

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%)

(parenthesis indicate size with data structures for nodal renumbering)

Mesh Information Cost

Total mesh storage/number of nodes (words/node)

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

Solution Process Information Cost

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	$2n$	$585n^2$	$132n^2$

n = #d.o.f per node

Solution: values of degrees of freedom

Element Matrices: unassembled element matrices

Global Stiffness: assembled, compressed row storage

Is Mesh Storage Significant?

Example

- 3-d elasticity problem, quadratic tetrahedral elements, iterative solver using compressed row storage for global stiffness.
  storage for solution process 375 words/node
  (6 words/node for solution, 369 words/node for global matrix)

  Classic mesh data structure adds 15 words/node, total = 390 words/node
  Largest hierarchic data structure (one-level representation) adds 31 words/node, total = 406 words/node
  4% difference in total storage
  Same problem with hexahedral elements results in 1% difference

Extra storage for hierarchic data structure does not seem significant

Comparison of Hierarchic Data Structures for Adaptive Analysis

$$M^3 \leftarrow\!\!-\,- M^2_{boundary} \leftarrow\!\!-\,-\!\!\rightarrow M^1 \rightleftarrows M^0$$

Ref: R. Biswas and R. Strawn, AIAA-93-0672

- Storage $\approx 22.5 N_M^3$ (not counting boundary information)
- Faces only represented on boundary
- Tailored for an edge-based refinement procedure

$$M^3 \rightleftarrows M^2 \longrightarrow M^1 \longrightarrow M^0$$

Ref: Y. Kallinderis and P. Vijayan, AIAA Journal 31(8), 1993

- Storage $\approx 27 N_M^3$
- Fast retrieval of downward adjacencies
- Some relations cannot be found without global searching

SCOREC Mesh Database

- Generic database for mesh information
- Mesh represented as a hierarchy of topological entities
- All information is accessed through set of operators (callable from C/C++ and Fortran)
- Common mesh representation allows various codes to be developed separately and then work together
- Object-oriented design, written in C (also a C++ implementation)
- All user interactions are through a set of operators that act on the objects in the database
- Bindings to other language provided by wrapper functions around native C functions
- User must give downward adjacency and classification - upward adjacency and all use structures are automatically created by mesh database
- Both dynamic and static versions of objects in database allow optimization for these different situations
- Operators constructed from core routines that provide next level adjacency and classification information

Database Modes

- Database can be operated in three modes:
  Static, minimal use
  Static, no entity uses
  Dynamic, no entity uses
  Static mode - only queries allowed
  Dynamic mode - queries and modifications allowed
- Internal representation of data varies between modes
- User interface remains the same for all modes

Implementation - Core Routines

Core routines depend on internal representation of data

- Next-level adjacency
  R_faces - get faces bounding region
  F_regions - get regions using face
  F_edges - get edges bounding face
  E_faces - get faces using edge
  E_vertices - get vertices bounding edge
  V_edges - get edges using vertex

- Classification
  EN_whatIn - get classification of entity

Higher level routines

Independent of internal representation of data - call core routines to access data.

Examples:

- Multilevel adjacency
  R_edges - retrieve edges bounding region
  R_vertices - retrieve vertices bounding region
  etc.
- Other queries
  E_otherFace - get the other face using an edge connected to a given mesh region
  F_inClosure, E_inClosure, V_inClosure - determine whether an entity is in the closure of another entity

Model Operators

Model - topological representation of either a F.E. Mesh or a geometric model

- Information retrieval
  M_nRegion, M_nFace, etc. - return the number of the given entities in the model
  M_nextRegion, M_nextFace, etc. - sequentially return each of the given entity in the model
- Modification
  M_addRegion, M_addFace, etc. - add an entity of the given type to the model
  M_removeRegion, M_removeFace, etc. - remove an entity of the given type from the model