Engineering with Computers (2004) 20: 286–293
DOI 10.1007/s00366-004-0296-0

ORIGINAL ARTICLE

Timothy J. Tautges

MOAB-SD: integrated structured and unstructured mesh representation

Received: 28 July 2003 / Accepted: 12 April 2004 / Published online: 20 August 2004

Abstract Structured and semi-structured (a.k.a. swept or extruded) hexahedral meshes are used in many types of engineering analysis. In finite element analysis, regions of structured and semi-structured mesh are often connected in an unstructured manner, preventing the use of a globally consistent parametric space to represent these meshes. This paper describes a method for mapping between the parametric spaces of such regions, and methods for representing these regions and interfaces between them. Using these methods, a 57% reduction in mesh storage cost is demonstrated, without loss of any information. These methods have been implemented in the MOAB mesh database component, which provides access to these meshes from both structured and unstructured functions. The total cost for representing structured mesh in MOAB is less than 25 MB per million elements using double-precision vertex coordinates; this is only slightly larger than the space required to store vertex coordinates alone.

Keywords Structured · Semi-structured · Mesh · Finite difference · Finite element

1 Introduction

The data structures used to represent a grid or mesh depend heavily on the type of mesh being represented. Three types of mesh can be described with respect to the storage methods used: structured, unstructured, and polyhedral. A structured mesh is one with an implied d-parametric space, where d is the topological dimension of the mesh (an alternative definition of a structured mesh is one where every interior vertex has exactly 2D neighbors). An unstructured mesh is one with or without structure. Unstructured mesh can be composed of hexahedra, tetrahedral, or a combination of those and other elements, and can be considered as a superset of the structured mesh type. Polyhedral meshes are composed of elements with an arbitrary number of bounding facets; these meshes are not discussed further in this paper.

Storing a mesh implies storing a description of the vertices and elements which make up the mesh. Vertices are usually described by their spatial position, while elements are described by their ''connectivity'', that is, by the vertices which form the element. However, for structured meshes, the connectivity of any element can be inferred by its parametric coordinates in a d-dimensional parametric space. Computer codes based on a structured mesh almost always make use of this optimization to reduce storage costs for the mesh. A structured mesh can be ''fully structured'', where elements are mapped into a single global parametric space, or ''block-structured'', where blocks of mesh each have their own parametric space. For block-structured mesh, vertices and elements at block interfaces are usually duplicated and matched together logically.

A 3D region of swept mesh consists of a group of unstructured quadrilaterals swept into multiple layers orthogonal to those quadrilaterals. While these meshes are currently always stored in an unstructured representation, the connectivity of successive layers of quads is identical. This structure can be used to greatly reduce storage costs.

This paper describes a technique for representing block-structured and swept meshes without using vertex/element duplication at shared block or sweep boundaries. Instead, vertices and elements on block interfaces are referenced in each block by mapping between the parametric spaces in the respective blocks. This technique can be used to map between the d-dimensional blocks, and

Sandia is a multiprogram laboratory operated by Sandia corporation, a Lockheed Martin company, for the United States department of energy under contract DE-AC04-94AL85000. This work was performed under the auspices of the U.S. Department of Energy, Office of Advanced Scientific Computing, SciDAL program.

T. J. Tautges
Sandia National Laboratories, Albuquerque, NM, USA
E-mail: tjtautg@sandia.gov

also to map to and from lower dimensions. Representing a structured mesh in this way provides a more natural link to the BREP-based geometry representation used in most modern mesh generation codes. This technique also supports the parameterization of extruded or swept unstructured hexahedral meshes. Since most unstructured hexahedral meshes used in practice are composed of structured and swept regions, the techniques described in this paper can be used to eliminate most of the storage associated with mesh connectivity in these meshes. For large meshes, the memory savings can be substantial.

After deriving and implementing the material described in this paper, we became aware of a similar technique used for file-based storage of structured mesh—the CFD general notation system (CGNS) [1]. A brief inspection of CGNS documents indicates that the underlying method of mapping between parametric spaces using integer-based transforms (described in the next section) is essentially the same. However, our work is more flexible because it allows the representation and mapping between blocks and interfaces of a lower dimension, and also treats semi-structured mesh, which is not addressed in CGNS.

This paper is organized as follows. Section 2 gives some background on structured mesh representation. Section 3 describes our technique for mapping between parametric spaces sharing an interface. Section 4 describes homogeneous transforms and their application to this problem. Section 5 describes the implementation of this technique in MOAB. Section 6 discusses other issues related to this problem and its implementation. Conclusions are given in Section 7.

2 Background

A structured mesh is one which fits into a d-parametric space, where d is the topological dimension of the mesh. A block of structured mesh can be either rectangular (i.e., it has four logical sides), or non-rectangular (arbitrary numbers of logical sides). We restrict ourselves in this paper to rectangular blocks of structured mesh; using the techniques described in this paper, non-rectangular blocks can be built from rectangular blocks with a small amount of extra data (which scales in the number of blocks).

There are other codes which represent structured mesh in this or other ways. For example, Overture [2] represents the parametric space using minimum and maximum parameter values, which it refers to as a block's indexRange. Multiple blocks share interfaces by duplicating the points in each sharing block and interpolating field values between them. The CHOMBO package [3] uses a similar means for constructing rectangular parameter spaces, which it implements in its Box class. Multiple rectangular blocks are united into a (possibly non-rectangular) domain, called a ProblemDomain, defined with a single global parametric space. Similarly, the GrACE package [4] constructs a hierarchical structured mesh by combining rectangular regions in a global parametric space. None of these codes treat semi-structured mesh or shared interfaces between blocks in a block-structured mesh.

On the unstructured mesh generation side, several unstructured hex meshing codes (e.g., CUBIT [5] and GAMBIT [6]) construct meshes by combining regions of swept and structured mesh. Until now, little has been done to take advantage of that structure in the representation of the mesh. Two things motivate the present work to take advantage of that structure:

1. Virtually all large, unstructured meshes used in practice are constructed from regions of semi-structured and structured mesh. This is because we lack a robust all-hexahedral meshing algorithm.
2. Both semi-structured and structured meshes are bounded in the next lower dimension by a structured mesh [7, 8], and the "side" boundaries of a semi-structured mesh admit a globally consistent parametric space [8]. Therefore, it is possible to map boundaries between structured and (side faces of) semi-structured regions into the parametric spaces of those regions.

These two points imply that techniques to optimize the storage of semi-structured and structured mesh would apply to almost all of current unstructured hex meshes.

One other item important to representing mesh is the notion of a "handle". A handle is simply a reference to an entity in a mesh, with the property that handles can be incremented and decremented to get other entities. There are many options for representing handles, e.g., as pointers or integers (which can both be incremented and decremented). The MOAB mesh database [9] uses unsigned long integers to represent handles, in part, so that contiguous ranges of handles can be represented in a constant-size datum, called a range. Computing the handle of a vertex or element in a block of mesh is a frequently used operation in most applications which use structured or semi-structured mesh.

3 Mesh parametric spaces

This section describes parametric spaces of structured and semi-structured meshes. The vertices and elements of a structured mesh region are stored as a contiguous block of entities, described by a start handle and the number of entities. This section describes the mapping between parametric coordinates for a given entity and the corresponding entity handle used to reference that entity. Mapping between such spaces at shared interfaces is discussed in the following section.

3.1 Structured mesh

A 3D, rectangular block of structured mesh is shown in Fig. 1. A block consists of vertices and elements. We describe a structured block's parametric space using

minimum and maximum parameter values $i_{\min}$, $i_{\max}$, $j_{\min}$, $j_{\max}$, $k_{\min}$, and $k_{\max}$, where i, j, and k are integers, and vertices neighboring each other in a given direction differ by one (this requirement could be relaxed; this is discussed in Section 4). By convention, the minimum parameter for elements is the same as for the vertices, while the maximum parameter is one less in each coordinate than for vertices. The parameterization technique in Fig. 1 also works for other dimensions, e.g., for two dimensions by removing k or for one dimension by removing j and k.

Given the parameterization convention in Fig. 1, along with values for the start handle for vertices and elements in a block, H_{vo} and H_{eo}, it is easy to compute the following quantities:

– Number of vertices in the block, N_{v}:

$$N_{\mathrm{v}} = (i_{\max} - i_{\min} + 1)(j_{\max} - j_{\min} + 1)(k_{\max} - k_{\min} + 1) \tag{1}$$

– Number of elements in the block, N_{e}:

$$N_{\mathrm{e}} = (i_{\max} - i_{\min})(j_{\max} - j_{\min})(k_{\max} - k_{\min}) \tag{2}$$

– Handle of vertex/element with parameters (i, j, k), $H_{\mathrm{v}}/H_{\mathrm{e}}$:

$$\begin{aligned} H_{\mathrm{v}} &= H_{\mathrm{vo}} + (i - i_{\min}) + (j - j_{\min})I + (k - k_{\min})IJ \\ H_{\mathrm{e}} &= H_{\mathrm{eo}} + (i - i_{\min}) + (j - j_{\min})I' + (k - k_{\min})I'J' \\ I &= i_{\max} - i_{\min} + 1 \\ I' &= i_{\max} - i_{\min} \\ J, J' &\text{ similar} \end{aligned} \tag{3}$$

– Connectivity of element with parameters (i, j, k), $C_{\mathrm{e},ijk}$:

$$\begin{aligned} C_{\mathrm{e},ijk} = \Big\{ & H_{\mathrm{v},ijk}, H_{\mathrm{v},i+1jk}, H_{\mathrm{v},i+1j+1k}, H_{\mathrm{v},ij+1k}, H_{\mathrm{v},ijk+1}, \\ & H_{\mathrm{v},i+1jk+1},\ H_{\mathrm{v},i+1j+1k+1}, H_{\mathrm{v},ij+1k+1} \Big\} \end{aligned} \tag{4}$$

Note that these relations are easily derived from a three-parameter structured index system, and that this has been done in many other previous efforts, in the area of structured mesh (see e.g., [3]) as well as for other applications. Also, these relations have to be modified slightly in the case of periodic mesh in one or more directions.

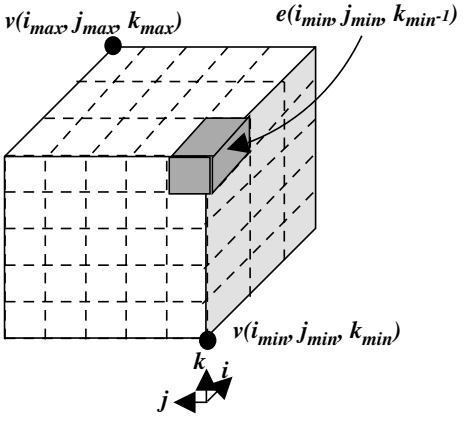

Fig. 1 Structured mesh with $d=3$ and parameter bounds of ($i_{\min}$, $j_{\min}$, $k_{\min}$)–($i_{\max}$, $j_{\max}$, $k_{\max}$) (vertices), ($i_{\min}$, $j_{\min}$, $k_{\min}$)–($i_{\max}-1$, $j_{\max}-1$, $k_{\max}-1$) (elements). Minimum and maximum vertices are indicated; sample element is shown with its parametric coordinates

3.2 Semi-structured mesh

The parameter space for a semi-structured, or swept, mesh, is shown in Fig. 2. Swept meshes can be parameterized in three parameters, i, j, and k. As before, the k-parameter specifies the vertex and element parameters in one direction, with vertices and elements having parameter ranges $k_{\min}$ to $k_{\max}$, and $k_{\min}$ to $k_{\max}-1$, respectively. However, for semi-structured mesh, the i and j parameters take on different meanings than for structured mesh. The i-parameter is used to parameterize the vertices in the cross-section, while j is used to parameterize the elements. This is necessary because in the unstructured (quadrilateral) mesh of the cross-section, there isn't a consistent way of parameterizing both vertices and elements with the same parameters.

The vertices in a semi-structured mesh are parameterized in i as follows. It was shown in [8] that the sides of a swept mesh are structured and periodic; that is, they form a loop which is closed in one parametric direction. We parameterize the mesh in that direction with the parameter range $i_{\min}$ to $i_{\max,0}$. Note, however, that because of the periodicity, the vertices normally found at $i_{\max,0}$ are really those at $i_{\min}$. For swept

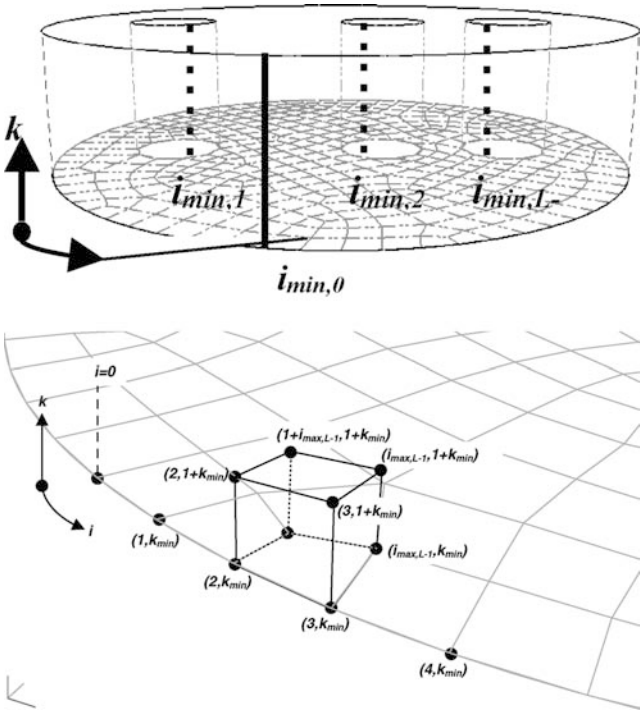

Fig. 2 Semi-structured mesh with $d=3$ and $i_{\max,l}$ as shown (*top*); one element in first layer (*bottom*), with connectivity $C_{\mathrm{e},jk}$ consisting of vertices with (i, k) parameters shown [parameters for hidden vertex are ($1+i_{\max}$, $L-1$, $k_{\min}$)]

meshes whose cross-section is not simply connected, there is more than one loop of side surfaces; this is the reason for the additional subscript on $i_{\max,0}$. Vertices on additional loops $l=1...\ L-1$ are parameterized in the range $i_{\max,l-1}$ to $i_{\max,l}$, where again vertices normally found at $i_{\max,l}$ are really those at $i_{\max,l-1}$. The vertices on the interior of the quadrilateral mesh are parameterized in the range $i_{\max,L-1}$ to $i_{\max}$, where this time there is a vertex at $i_{\max}$.

By re-using parameters $i_{\max,0},\ldots,\ i_{\max,L-1}$, all the vertices in the cross-section are contiguous in i. This allows us to store the vertices in a contiguous range of handles, and compute the handles directly from the block parameterization plus the block starting vertex handle. This promotes efficiency in both memory and time.

The j-parameter is used for quadrilateral elements in the cross-section; since these elements are unstructured, they must be represented with a connectivity array:

$$C_{\mathrm{q}} = \left\{ \{i_a, i_b, i_c, i_d\}_j,\ j = j_{\min} \ldots j_{\max} \right\}$$

where i_a, i_b, i_c, and i_d are the i parameters of vertices which form element j. Since every layer of quadrilaterals in the sweep has the same connectivity, the storage for this array is amortized over all the layers in the sweep. Hexahedral element connectivity can be inferred from that array plus the j and k parameters for that element, as described below.

In addition to the bounds on i, j, and k, the parameterization for semi-structured meshes includes an L-dimensional vector $i_{\max,l}$, $l=0...L-1$, where L is the number of loops in the cross-section. Thus, the data required to represent the connectivity of a semi-structured mesh is as follows:

- Parameter min/max $(i_{\min}, j_{\min}, k_{\min})$, $(i_{\max}, j_{\max}, k_{\max})$
- Cross-section element connectivity vector, $C_{\mathrm{q},j} = \{\{i_a, i_b, i_c, i_d\}_j, j=j_{\min}\ldots j_{\max}\}$
- Loop-wise ending parameter vector $i_{\max,l}$, $l=0...\ L-1$
- Number of loops, L
- Vertex, element start handles H_{vo}, H_{eo}

Using these data, the following quantities can be computed:

- Number of vertices in the block, N_{v}:

$$N_{\mathrm{v}} = (i_{\max} - i_{\min} + 1)(k_{\max} - k_{\min} + 1) \tag{5}$$

- Number of elements in the block, N_{e}:

$$N_{\mathrm{e}} = (j_{\max} - j_{\min} + 1)(k_{\max} - k_{\min}) \tag{6}$$

- Handle of vertex/element with parameters (i, j, k), H_{v}/H_{e}:

$$\begin{aligned} H_{\mathrm{v}} &= H_{\mathrm{vo}} + (i - i_{\min}) + (k - k_{\min})I \\ H_{\mathrm{e}} &= H_{\mathrm{eo}} + (j - j_{\min}) + (k - k_{\min})J \\ I &= i_{\max} - i_{\min} + 1 \\ J &= j_{\max} - j_{\min} + 1 \end{aligned} \tag{7}$$

- Connectivity of element with parameters (i, j, k), C_{ijk}:

$$\begin{aligned} C_{\mathrm{e},jk} &= \left\{ \left\{ H_{\mathrm{v}}\left(C_{\mathrm{q},jk}\right) \right\}, \left\{ H_{\mathrm{v}}\left(C_{\mathrm{q},jk+1}\right) \right\} \right\} \\ &= \left\{ H_{\mathrm{v}} \begin{pmatrix} (i_{j,a},k), (i_{j,b},k), (i_{j,c_1},k), (i_{j,d_1},k), \\ (i_{j,a},k+1), (i_{j,b},k+1), (i_{j,c},k+1), (i_{j,d},k+1) \end{pmatrix} \right\} \end{aligned} \tag{8}$$

To our knowledge, this technique for parameterizing semi-structured meshes has not been used before.

The parameterization for a block of vertices and elements, as described in this section, applies to the block as a whole, which may include patches of vertices and elements (of one lower dimension) from other blocks. The handles for these entities cannot be computed using the relations above, because they are stored with respect to another parametric space. However, by mapping between the parametric spaces of the two regions sharing the boundary, the handles for shared entities with respect to the native parametric space in which they are stored can be found. This procedure is described in the following section.

4 Homogeneous transforms

Our goal is to store blocks of mesh using implicit connectivity, which we accomplish using a parameterization for vertices and elements in the block. However, many of the problems of interest, and all those containing semi-structured mesh, do not admit a consistent global parameterization across all blocks. In this situation, multiple parameterizations must be used. By definition, the interface between two blocks of mesh can be described in the parameter space of either block, or in its own parameter space. In this section, we describe how to map between these various parametric spaces.

As an example, consider the blocks in Fig. 3. Each block is a structured mesh (not shown), and the two blocks share a 2D structured mesh. The blocks each have their own parametric space, (i, j, k) and (i', j', k'). Because the 2D mesh is structured, it can also have its own (2D) parametric space, (i'', j''). There must exist a map between these three systems because the interface exists in all three blocks simultaneously.

4.1 Transforms between coordinate systems

To map between these spaces, we borrow the notion of mapping between coordinate systems, a common practice in geometric modeling [10]. Given a point with coordinates p and q in parametric spaces (i, j, k) and (i', j', k'), respectively, the two coordinate vectors are related by:

$$q = (p\mathbf{R} + t)s \tag{9}$$

where $\mathbf{R}$ is a rotation matrix, and t and s are translation and scaling vectors, respectively. $\mathbf{R}$, t, and s can be

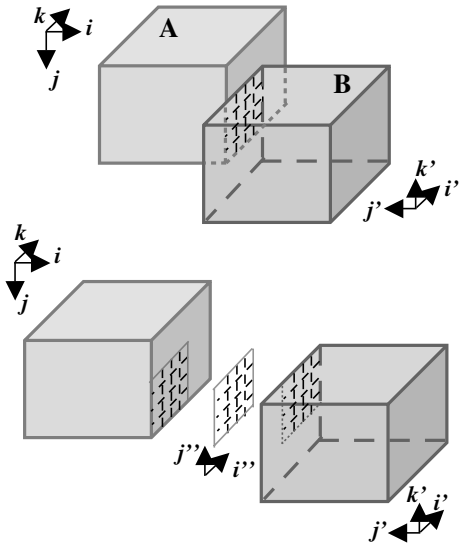

Fig. 3 Two structured blocks of mesh sharing an interface, where each block and the interface can have separate parameter spaces

combined into a single 4×4 matrix, commonly referred to as the *homogeneous transform* matrix **M**, where:

$$\begin{aligned} \mathbf{M} &= \begin{pmatrix} \mathbf{R}s & 0 \\ t & 1 \end{pmatrix} \\ p &= (i, j, k, 1) \\ q &= p\mathbf{M} \end{aligned} \tag{10}$$

A sequence of transformations is represented as the product of the homogeneous transform matrices. Because **R** is orthogonal, its inverse is its transpose; combined with the special structure of **M**, the inverse is easily computed:

$$\begin{aligned} \mathbf{M}^{-1} &= \begin{pmatrix} (\mathbf{R}s)^{\mathrm{T}} & 0 \\ -t(\mathbf{R}s)^{\mathrm{T}} & 1 \end{pmatrix} \\ p &= q\mathbf{M}^{-1} \end{aligned} \tag{11}$$

Finally, because parameters in a structured mesh are integers and are always logically orthogonal, the elements of the rotation and translation matrix/vector are also integers. This means that parametric transforms are done using integer arithmetic, which is much faster than floating point operations.

4.2 Structured block with referenced interfaces

Building a structured block of mesh consists of designating a parametric space for the elements, and referencing blocks of vertices into that space such that all parameter combinations within the block of elements are occupied by exactly one vertex. For example, the mesh shown in Fig. 4 contains four regions of vertices (distinguished in the figure by different shadings) and each vertex in the mesh comes from one of those. Note that vertex regions need not be only on the boundary of the region, but they must be rectangular.

In general, the larger the blocks of vertices and elements, the less storage required to represent the mesh (because this minimizes auxiliary data such as parameter space bounds and mapping transformations between blocks). The vertex referencing method described above is flexible in that vertex blocks of dimension d can be referenced from larger blocks of any dimension greater than or equal to d. For example, the lower boundary vertex block, which has $d=1$, can be referenced from blocks of $d=1$, 2, or 3. This means that $3d$ blocks of structured mesh can share interfaces of $d=0$, 1, or 2 *without ever explicitly representing those interfaces outside the 3d blocks*. Alternatively, a lower-dimensional interface can be constructed by referencing vertices completely from higher-dimensional blocks. Explicit representation of lower-dimensional boundaries is necessary in some cases, for example, when the lower-dimensional boundary is associated with an entity in a geometric model. Referencing the vertices from higher-dimensional blocks allows those vertices to be stored in larger blocks, which reduces storage costs.

Semi-structured mesh works the same way as the structured examples given above. However, there will be at least one semi-structured vertex block, corresponding to the cross section of the sweep, i.e., the unstructured quadrilateral mesh.

4.3 Construction

There are many ways to construct a structured or semi-structured mesh represented using the techniques described above. In the simplest case, a single block can be constructed from one block of vertices. A more interesting example is the case where one starts with a group of regions which share interfaces and which are all to be meshed with structured or semi-structured blocks of elements. If we assume that each d-dimensional region can identify any lower-dimensional regions already containing a mesh, and each meshed region can identify its origin and the next vertex in each coordinate direction, then the current region's mesh can be constructed as follows:

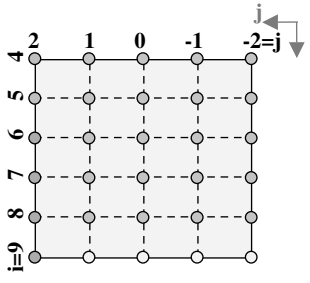

Fig. 4 Structured mesh containing four regions of referenced vertices. For this mesh, $(i_{\min}, j_{\min}) = (4, -2)$, $(i_{\max}, j_{\max}) = (9, 2)$, and the origin is in the upper right corner of the mesh

1. Compute a parameterization and find the location of (i_{min}, j_{min}, k_{min}) (structured) or (i_{min}, k_{min}) and $i_{max,l}$, $l=0...\ L-1$ (semi-structured) for the current region.
2. For each region of referenced vertices, find vertices in the current region corresponding to the origin and other vertices in the referenced region, and their parameters in both regions. Use these to compute the homogeneous transform matrix, using the standard three-point rule [10].
3. (Optional) Replace any lower-dimensional vertex blocks with references to vertices in current block (this improves storage efficiency by increasing average size of vertex blocks).
4. Construct any missing vertex blocks to fill the current region's vertex parameter space.
5. If block is to be semi-structured, construct the connectivity array $C_{q,j}$ based on $i_{max,L-1}... i_{max}$ computed in step 1. Store with block of elements constructed in step 6.
6. Allocate a block of elements/handles which represents the elements in the current block, storing the block's auxiliary data with the block.

5 Implementation

The methods described above have been implemented in the MOAB-SD (MOAB structured data) component of the MOAB mesh database library [9]. MOAB is a library and API for representing mesh which is both space- and time-efficient, allowing construction and storage of large (>1-million element) meshes. MOAB has been constructed as part of the Terascale Simulation Tools and Technologies (TSTT) center [11], supported by DOE's SciDAC program [12]. MOAB also provides an implementation of the TSTT mesh interface, which promotes interoperability between various software components which generate or use mesh data.

MOAB uses separate mechanisms for representing and grouping mesh entities. Sequences are used for representing contiguous handle ranges of vertices and elements. This representation is unique, such that a given entity handle is represented in exactly one sequence. Entity sets are used for arbitrary groupings of entities, including other sets; they are used to represent the various types of groupings typically found in meshes, for example, those representing boundary conditions, geometric topology association, or inter-processor interfaces.

MOAB-SD implements structured blocks of vertices and elements using the classes ScdVertexSeq and ScdElementSeq, respectively, which are derived from normal vertex and element sequence classes in MOAB. An additional class, VertexSeqRef, is used to store data associated with a vertex block referenced from a ScdElementSeq. The ScdVertexSeq class stores a start vertex handle; the min/max parameters of the vertex block; and, for convenience, the number of vertices and elements in each parametric direction (which could otherwise be computed from the min/max parameters). ScdElementSeq stores the start element handle; the min/max parameters of the block of elements; the numbers of vertices and elements in each parameter direction; and a list of VertexSeqRef objects. Each VertexSeqRef in this list represents a reference of a vertex block into a block of elements. This class stores the homogeneous transform matrix for mapping between the parameter spaces of the referenced vertex block and the ScdElementSeq, and its inverse; the min/max parameters of the referenced vertex block with respect to the ScdElementSeq; and a pointer to a ScdVertexSeq instance being referenced.

ScdVertexSeq and ScdElementSeq inherit from the classes representing normal vertex and element sequences in MOAB (VertexEntitySequence and ElementEntitySequence, respectively). ScdVertexSeq has all the data associated with a VertexEntitySequence, including vertex spatial positions. Thus, ScdVertexSeq has a size slightly larger than that of a corresponding VertexEntitySequence representing the same mesh. ScdElementSeq, on the other hand, has no connectivity information, in contrast to ElementEntitySequence, making it much smaller.

The expected memory savings due to not explicitly representing element connectivity can be computed as follows: the minimum information required to store a mesh is to store vertex coordinates and element connectivity. Assuming double precision storage of coordinates and 32-bit representation of vertex handles (using either pointers or some type of index to reference vertices), a hexahedral mesh can be represented in $24N_v + 32N_e$ bytes, where N_v and N_e are the number of vertices and elements, respectively. For a single square block of elements with l elements on each side, N_v approaches l^2 with increasing l, and N_e is equal to l^2. Therefore, asymptotically, the storage of a structured block of N_e elements is approximately $56N_e$ bytes. Eliminating the representation of connectivity information reduces this to $24N_e$ bytes, a savings of $32/56 = 57.1\%$. Thus, asymptotically, the same (large) mesh represented using ScdVertexSeq and ScdElementSeq will require 57% less storage space than the equivalent representation in VertexEntitySequence and ElementEntitySequence.

The above techniques were implemented in MOAB, and tested by initializing and querying a single square block of hexahedral elements. This was compared to the cost of creating each element individually and as a block in MOAB, and to creating this mesh using the data structures from CUBIT. All tests were performed on a 1.2 GHz Pentium III-based computer with 1.0 GB of RAM running Linux.

The memory usage results are shown in Fig. 5. Memory savings due to using structured representation exceed 50%, as expected. Memory savings compared to data representation in CUBIT is even greater, due to extra information (entity IDs and pointers to owning geometry) stored with mesh entities in CUBIT. The data in Fig. 5 shows that MOAB can store structured mesh at

a cost of less than 25 MB per million elements, and can store 40 million elements per GB of RAM.

The CPU times required to initialize and query this mesh are shown in Figs. 6 and 7, respectively. Initialization of the structured representation takes substantially less time, because it requires only the computation of a few parameter bounds and initialization of very little memory, compared to the other representations. However, query time for the structured representation is substantially more. Computing the connectivity for structured meshes requires more time than simply copying that information from memory. While the time difference appears substantial, it is small compared to other operations typically being performed on the mesh. Therefore, we view that as an acceptable tradeoff.

Storing semi-structured meshes using the techniques described here has not yet been implemented. However, because these meshes use much of the same code as for structured meshes for initializing bounding mesh referenced from other regions, it will be a relatively small effort to add support for these meshes. Empirically, the number of vertices and elements in a swept mesh are approximately equal. Therefore, we expect a memory saving of approximately 57% for these meshes as well.

6 Other issues

Structured mesh is not strictly defined only for rectangular regions. The CUBIT code [5] implements the so-called ''submap'' algorithm [7], which generates structured mesh on other types of regions in 2D and 3D. Since the algorithms described here only treat rectangular regions, submapped regions cannot be represented in a single sequence. However, these regions can easily be decomposed into rectangular regions. Also, in this case, the transform between parametric spaces is trivial because the rectangular regions can be put into the (single) parameter space of the submapped region. In our implementation, a submapped surface or volume is represented using an entity set, which includes entities in multiple sequences.

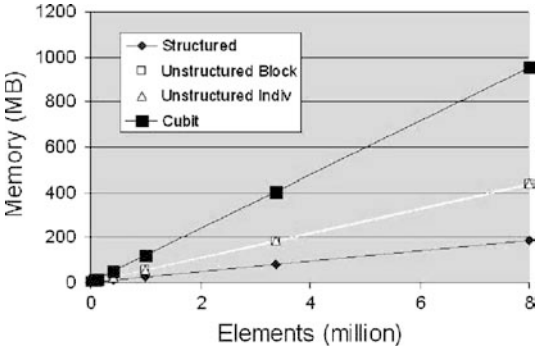

Fig. 5 Memory usage for MOAB structured representation, compared to MOAB unstructured mesh (created individually and as a block) and CUBIT-based unstructured mesh

Likewise, sweep meshing is not restricted to extruding one surface mesh to one target mesh. Multiple surfaces can be swept, varying numbers of layers to multiple target surfaces [13, 14]. In this case, the composite of all side surfaces will be submapped. However, as with submapped regions, multisweep regions can be decomposed into and, therefore, represented using a group of semi-structured sequences of single source and target surfaces. In fact, several implementations of the multisweep algorithm actually generate the mesh in this way anyway.

One interesting problem not addressed here is that of verifying that the vertex parameter space of a region is completely ''covered'' by all referenced vertex regions. Currently, a ''brute force'' method of iterating over all parameter values is used. However, since the referenced regions and the space being covered are all rectangular, and are represented with min/max corners of parametric coordinates, there is probably an efficient algorithm for verifying that covering based on min/max coordinates.

7 Conclusions and future work

The algorithms and techniques described in this paper provide a representation of structured and semi-struc-

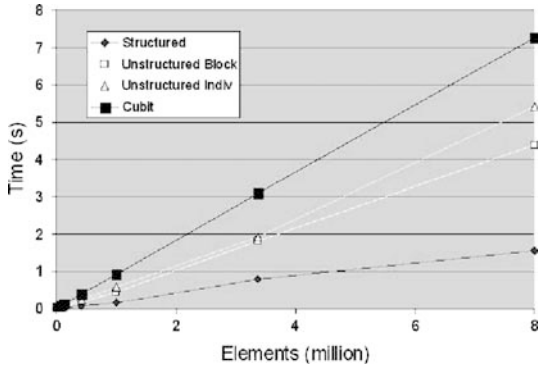

Fig. 6 CPU time for constructing MOAB structured representation, compared to MOAB unstructured mesh (created individually and as a block) and CUBIT-based unstructured mesh

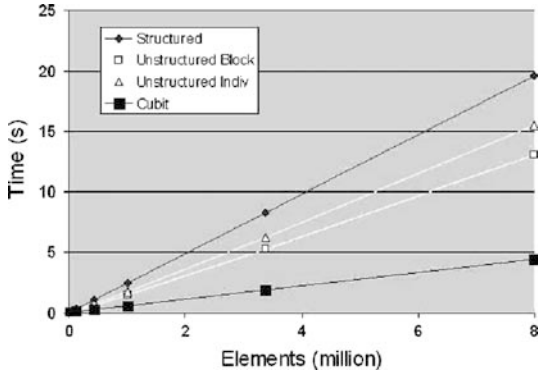

Fig. 7 CPU time for constructing MOAB structured representation, compared to MOAB unstructured mesh (created individually and as a block) and CUBIT-based unstructured mesh

tured (i.e., swept) mesh which is efficient in both storage and evaluation cost. Using implicit representation of element connectivity results in a 57% reduction in asymptotic minimum storage cost for structured mesh, and is expected to be similar for semi-structured mesh. Structured mesh representation using these methods has been implemented in the MOAB-SD mesh database library.

The capability to read structured mesh data and initialize these data in MOAB-SD will be developed soon. This will allow applications to operate on these meshes using both structured and unstructured queries. Mesh structure is one example of meta-data which exists when mesh is generated, but is currently not made available to downstream applications. The representation of other types of meta-data are also being incorporated into MOAB, including representation of geometric model topology, boundary condition groupings, and interprocessor interfaces. We plan to investigate the use of these meta-data in downstream applications in the future.

Experience with large meshes has shown that reducing mesh storage costs by 57% will not be sufficient. Therefore, other methods for reducing mesh storage costs will be needed. Possible techniques which may bring those reductions include out-of-core storage, in-memory compression of coordinate data, or implicit representation of coordinate data. Of course, spreading the representation of a large mesh over parallel processors is also an effective technique; this is already being explored in the context of parallel mesh generation, and will probably be the most immediate solution for representing very large meshes.

Acknowledgements The author wishes to acknowledge Karl Merkley, Ray Meyers, Clint Stimpson, and Corey Ernst at Elemental Technologies, Inc., for their great work implementing other parts of MOAB, on which this work is based. We also wish to acknowledge Mike Heroux of Sandia National Laboratories for his quick derivation of the inverse transform matrix described in Section 4.

References

1. CFD general notation system standard interface data structures, document ver 2.3.5. Draft AIAA recommended practice R-101A-200X. Available at http://www.grc.nasa.gov/WWW/cgns/sids/sids.pdf
2. Brown DL (2002) Overture software for solving PDEs in complex geometry. In: A workshop on the ACTS collection, Berkeley, California, September 2002, UCRL-PRES-150012
3. Chombo: adaptive mesh refinement library. Available at http://www.seesar.lbl.gov/anag/chombo/
4. Parashar M, Browne JC (2000) Systems engineering for high performance computing software: the HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement, IMA vol 117. Structured adaptive mesh refinement (SAMR) grid methods
5. Blacker TD et al (1994) CUBIT mesh generation environment, vol 1: user's manual', SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico. Available at http://sass1693.sandia.gov/cubit/release/doc-public/Cubit_UG-4.0.pdf
6. Gambit, Fluent Inc. http://www.fluent.com/software/gambit
7. White D, Mingwu L, Benzley S, Sjaardema G (1995) Automated hexahedral mesh generation by virtual decomposition. In: Proceedings of the 4th international meshing roundtable, Albuquerque, New Mexico, October 1995. Sandia National Laboratories, SAND95-2130
8. White DR, Tautges TJ (2000) Automatic scheme selection for toolkit hex meshing. Int J Numer Methods Eng 49:127–144
9. Tautges TJ, Meyers RE, Merkley K, Stimpson C, Ernst C (2004) MOAB: a mesh-oriented data base. Sandia National Laboratories report, Sandia National Laboratories, Albuquerque (to appear)
10. Mortenson ME (1985) Geometric modeling. Wiley, New York
11. The Terascale Simulation Tools and Technology (TSTT) Center. Home page at http://www.tstt-scidac.org/
12. SCIDAC: scientific discovery through advanced computing. Home page at http://www.scidac.org/
13. White D, Saigal S, Owen S (2003) CCsweep: automatic decomposition of multi-sweep volumes. In: Proceedings of the 4th symposium on trends in unstructured mesh generation, Albuquerque, New Mexico, July 2003
14. Lai M, Benzley SE, White DR (2000) Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. Int J Numer Methods Eng 49:261–275 (2000)