Ray J. Meyers, Karl Merkley, Timothy J. Tautges, "SNL Implementation of the TSTT Mesh Interface", 8th International Conference on Numerical Grid Generation in Computational Field Simulations, Honolulu, HA, June 2-6, 2002.

SNL Implementation of the TSTT Mesh Interface

Ray J. Meyers
Karl Merkley

Elemental Technologies, Inc.
American Fork, Utah
ray@elemtech.com
karl@elemtech.com

Timothy J. Tautges

[1]Sandia National Laboratories
Albuquerque, NM
University of Wisconsin-Madison
Madison, WI
tjtautg@sandia.gov

Abstract

The Terascale Scale Tools and Technologies Center of the U.S Department of Energy has defined an interface standard to be used by applications which need to access the mesh representation data associated with Finite Element analysis and simulation. Use of this interface promotes greater interoperability between diverse mesh data representations and the applications which access them.

This paper describes the TSTT Mesh Query Interface as currently defined, and describes an implementation of the interface created at Sandia National Laboratories. Implementation advantages and challenges are presented, along with preliminary data on some performance aspects of the interface.

Introduction

It is widely believed that mesh generation and other pre-processing technologies form a bottleneck in the simulation process, limiting the speed at which simulations can be performed for varying domain models. At the same time, many advanced techniques in computational analysis require enabling technology tools which are not available in a form which can easily be integrated into the analysis in question. Currently, the barriers to introducing a new tool in an existing pre-processing or analysis process are so large that developing custom tools is easier than using existing ones. These barriers must be reduced so that these analyses can take advantage of the most advanced enabling technology tools.

[1] 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.

There are many centers of expertise in pre-processing[1][2][3], with varying primary foci (in both pre-processing technologies as well as applications). However, there are also many areas of commonality between these efforts, including geometry and mesh representation, various meshing techniques, and adaptive methods. The ability to mix and match tools from these areas would be of direct benefit to applications, allowing them to choose the most appropriate tools for their particular needs.
The Terascale Simulation Tools & Technologies center[4] is part of a larger US DOE program named Scientific Discovery through Advanced Computing (SciDAC)[5]. The goal of the TSTT center is to develop interfaces and tools for meshing and other enabling technologies, and to modify the various tools which already exist at TSTT member sites to operate through these interfaces. These tools will then be integrated into various SciDAC applications, and any missing technologies developed to meet applications needs.

The most common form of data communicated during various points in the analysis process is that of mesh representation. The mesh represents the domain being analyzed, and provides the structure for arranging and storing analysis data. Interactions between almost any of the various enabling technology components will require a common interface for creating and accessing mesh data. Therefore, a common mesh interface is an important component in the TSTT effort.

The TSTT center has designed a preliminary specification of a mesh interface. This is an interface specification only; the actual representation and implementation of the interface is intentionally left out, allowing various implementations which have various benefits and costs. This paper describes one such implementation, along with the requirements, costs and benefits of this implementation.

Summary of the TSTT mesh interface

The TSTT mesh interface separates the mesh representation from the tools which use it, allowing both to act as components. This greatly simplifies the interchangeability of **both** the implementation and the tools which use it. The TSTT specification is an **interface** only. Many implementations of the interface may exist, each implemented independently. Each complete implementation should be interchangeable with other implementations. Specialized implementations can be created which are optimized for specific contexts; e.g. structured or swept meshes.

Access to data in the mesh database is provided using the concept of handles wherever possible. Handles provide an implementation independent way for the mesh data user to refer to mesh entities. The actual implementation and interpretation of the handle are left to the mesh interface implementation, allowing flexibility in the implementation while maintaining interchangeability between implementations.

The TSTT Mesh Interface is a relative new effort. Currently the working group is concentrating on providing an interface specification for the query portion of the mesh interface. At the time of this writing, this specification is still very much in flux as the TSTT members strive to provide a complete and usable interface. The interface itself is currently specified as a C++ pure virtual class. Bindings for other languages (e.g. C and Fortran) will be provided, possibly through the use of an interface specification language to provide interoperability support.

Supported Entities. The TSTT Mesh Interface provides for a hierarchy of entity types for describing a mesh:

- Mesh Vertex
- Mesh Edge
- Mesh Face
- Mesh Region

These are, respectively, zero, one, two, and three dimension entities. There is also the notion of sub-divisions or types within these, such as Mesh Face types TSTT_QUADRILATERAL and TSTT_TRIANGLE.

Mesh Services. The interface provides several general types of services:

1. <u>Handles to Entity Information</u>. Mesh entities which have been stored in the database can be retrieved. Currently, the interface allows the retrieval of either all the entities of given type, or retrieval of entities singly through an iterator. These handles can in turn be used through the interface to access all pertinent information about entities in the database.
2. <u>Adjacency information.</u> Any mesh entity can be queried for the adjacent topological entities of any other dimension. An example of this is a TSTT_REGION entity can request for all of the TSTT_FACE entities which bound that entity.
3. <u>Tag information.</u> Tag information is auxiliary information which can be attached to entities in the database or to the database as a whole. This tag information can be added, queried, reset, and destroyed as needed. An example might be adding a tag to a TSTT_FACE entity which holds the color which should be used to draw the entity.

SNL Implementation Requirements

Sandia's mission requires the use of advanced simulation across a wide range of physics, including large-deformation structural mechanics, heat transfer, and some fluid dynamics. Likewise, the format and origination of the domain models vary, often starting as CAD-based component designs but proceeding to highly deformed simulation models. Analysis codes used at Sandia include everything from low-end commercial tools, e.g. Ansys, all the way up to custom, highly parallelized analysis tools like Alegra[6] and Sierra[7]. In order to streamline the overall simulation process, enabling technologies should support this spectrum of uses in a seamless manner. Specific requirements for this environment are discussed in this section.

Comprehensive. Because of the variety of analyses and computational environments, our mesh interface implementation must be comprehensive in its treatment of the following items:

- Element type: generally restricted to the finite element "zoo", i.e. tetrahedral, hexahedra, wedges, pyramids, etc.
- Input/output format: both lab-specific formats (e.g. ExodusII[8]) and commercial (e.g. Ansys, Patran, etc.)
- Mesh size (up to 1B elements)
- Computational platform (PCs, workstations, massively parallel computers)
- Tools. The implementation must be applicable to a wide range of the common tools which operate in these environments on these types of meshes

Modular. In order to guarantee that any mesh created in one tool can be accessed in that same form from another tool, interface implementations in both tools must be functionally equivalent. For example, if the mesh representation which serves a mesh generator is able to represent billions of elements, any visualization tool which is to operate on those meshes must also have that capability. Conversely, we also would like to minimize the overall size and complexity of our tools; this requirement implies a careful grouping of functionality into distinct modules such that applications can choose to incorporate only a subset of the TSTT capabilities. For example, in applications like mesh generation, information about the geometric domain model is useful, whereas for evaluating many aspects of mesh quality the geometric model is unnecessary. Because of the variety of applications at SNL, we require that the mesh interface implementation be independent of the geometry. Note that this also allows applications to vary the mesh and geometry interface implementations independently.

Interoperable. One of the primary bottlenecks in the design to analysis process is the loss of information resulting from conversion of data between various representations. Lost data prevents effective feedback in the design to analysis loop. We strive for a mesh interface implementation which can operate on mesh data in its native form no matter what the source (commercial codes, national lab codes, etc.) or the destination (analysis code, visualization, etc.) This reduces the number of copies of a given mesh which must be kept and tracked through the analysis process.

Scalable. When discussing terascale computing, scalability is always an issue. Simply put, our mesh interface implementation must be capable of handling unstructured meshes in excess of 1 billion elements. In addition, this must be possible *with no loss of functionality on meshes of that size*. This can be a challenging task, when considering applications which for example need to reference interior nodes and faces of a 3D mesh.

MDB Implementation of the TSTT Interface

The MDB (Mesh DataBase) is an effort to create a next generation database for Sandia's finite element effort which better meets the unique Sandia requirements discussed above. MDB is scheduled to be a component of multiple tools; among the first are CUBIT[1], a mesh-generation code, and Verde [9], which analyzes element quality and topology of FE meshes. MDB will utilize the TSTT mesh interface to assist in packaging MDB functionality in a way which will maximize reuse and minimize maintenance. MDB will also be used in conjunction with other modules such as the Common Geometry Module [10], to demonstrate and further explore component interoperability, data hierarchies, etc.

To test some aspects of the TSTT mesh interface, we have incorporated it into the Verde quality verification tool. We did not create a new database implementation for this exercise; rather, we isolated the current Verde database from the rest of the code using the TSTT interface. This is a useful exercise for two reasons:

1. By using an existing product which was not written with the TSTT interface in mind, we hoped to "shake-out" the interface, testing for comprehensiveness and correctness of the TSTT specification.
2. By using an existing database, we could get data on the raw, worst-case cost of imposing the TSTT interface in place of the more distributed interface Verde was currently using.

Performance

To measure the performance impact of the TSTT interface on Verde, three versions of Verde were compiled:

1. The base version, an unchanged version of Verde 2.5.
2. A raw TSTT version, where the TSTT interface was interposed between the main Verde code and the Verde database code. No attempt was made to optimize the code as a result of introducing the TSTT interface.
3. An enhanced TSTT version, where both the calling code and the database server code were modified to make the TSTT interface more efficient to use. These changes were relatively minor, and no other changes were made to the code to improve efficiency

Two tests were run to provide benchmarks for execution time and memory usage. The first test was the regular regression test suite used for Verde quality control. This test suite performs approximately 40 tests; most of these are small models. Thus, startup and overhead associated with executing the program may unfairly dominate. The second test was of a fairly large model containing approximately 1.5 Million linear (8-node) hexes. These tests were chosen as typical uses of Verde; no attempt was made to exercise all portions of the interface or to balance usage in any way.

The following table summarizes the results for the tests. All tests were normalized relative to the performance of the original version.

	Test Suite		**1.5 M Hex Problem**	
Version	**Time**	**Memory**	**Time**	**Memory**
Original	1.0	1.0	1.0	1.0
Raw TSTT	1.06	1.0	1.10	1.0
Enhanced TSTT	1.04	1.0	1.07	1.0

Figure 1: Comparative Execution Times and Memory Requirements

The performance results were consistent with expectations. The difference of about three percent between the raw and enhanced versions is largely a measure of the degree of mismatch between the native Verde database and the TSTT interface. This inefficiency is not surprising considering that the Verde database was designed without any knowledge of the TSTT interface. The four to seven percent loss of efficiency between the original and the enhanced version appears to be largely due to some inherent inefficiencies in the TSTT interface to facilitate greater generality and language independence.

Interface Enhancements

There are several useful enhancements we have made to the current TSTT interface.

MeshSet: Arbitrary Grouping Mesh Entities

One of the goals of the TSTT center is to explore the natural data hierarchies which exist in the various simulation data representations, which include geometry, the initial mesh, partitioned mesh, and h-refined mesh. Currently, the information relating these levels to each other is either lost entirely or not used in most simulations. We have defined a class of entities called MeshSets which can be used to preserve these hierarchies and relationships between them.

We define a MeshSet as an arbitrary grouping of mesh entities (VERTEX, EDGE, etc.); in addition, MeshSets can hold other MeshSets. As for mesh entities, we allow tags to be defined and written on MeshSets. We also allow the embedding of parent/child relationships between MeshSets, which are used to represent hierarchical relationships between various groups of mesh entities. MeshSets impose no restrictions on the kind of entities contained or how many sets a given entity can appear in; rather, we assume that conventions will be defined which determine the types of data stored in particular sets. The functional interface for MeshSet is under development, and will be published with other MDB information when finalized.

In MDB MeshSets, the lists of entities and MeshSets are both stored in STL vector objects, sorted using the entity or MeshSet handle. This aspect of the implementation may change eventually, after performance measurements have been made under representative application conditions.

MeshSet Applications

We use MeshSets to store a wide variety of data relationships typically found in the analysis process; some of these are described next.

ExodusII Blocks, NodeSets, SideSets

Because we intend to use this implementation of the TSTT mesh interface to support mesh representation in Verde and CUBIT, it will be necessary to read and write ExodusII files into/from MDB. ExodusII defines the usual nodes/vertices and elements, and also defines three types of grouping objects: Element Block, which stores a group of elements; NodeSet, storing a group of nodes; and SideSet, which holds a group of element “sides” (each represented by an element number and a side number). This entities are used to avoid embedding in the mesh file information specific to a given analysis type (e.g. Temperature, Pressure, etc.).

Element Blocks and NodeSets map directly over to MeshSets, which, by convention, are given tags of type "ElementGroup" and "DirichletBC", respectively. SideSets are not as straightforward. While ExodusII represents sides as (element #, side #) pairs, we choose to represent sides as elements of lower dimension, and group them into a MeshSet. Mapping a SideSet to an MDB MeshSet requires creating an element for each side, then adding those elements to a MeshSet. By convention, we put a tag with name "NeumannBC" on these types of MeshSets.

In addition to tags identifying the type of BC MeshSet, other tags are written to these MeshSets to hold user-assigned name and id information, analogous to how users identify Blocks, NodeSets and SideSets in ExodusII. Optionally, tags could be used to store application-specific information about the actual type of boundary condition (Pressure=64.0Mpa, etc.); however, we assert that this information is better stored elsewhere and related back to the MeshSet using the name or id of the MeshSet.

Geometric Topology

A topic of increasing interest in the analysis community is using a geometric model in conjunction with the mesh for various analysis purposes. Some examples include projection to curved surfaces after adaptive mesh refinement[7], and smooth boundary conditions[11]. For these reasons, we store information in the mesh about its association to geometry, but in a manner which is independent of the actual representation of that geometry.

In order to unambiguously associate mesh to geometry, mesh elements of equal dimension to each geometry entity must be represented. To store this information, we create a MeshSet for each geometric entity containing the elements owned by that entity (vertex association can be inferred unambiguously from those data). The MeshSet for a given geometric entity are tagged with the name or id of the corresponding geometric entity, and with a tag type (we use "GeomDimension") and value (equal to the entity dimension) which identifies these MeshSets as corresponding to geometric entities.

The ability to traverse geometric topology defined for a mesh has also been requested, e.g. for re-defining vertices and/or elements included in boundary condition groups. This can be accomplished using "GeomDimension" MeshSets, by embedding parent/child relations between sets; this eliminates the expense of reading the geometric representation to get this information.

Other Applications

There are many other applications of MeshSets for solving specific analyst needs. MeshSets can represent groups of elements in (possibly) overlapping regions found in Chimera grids; the partition of a mesh for solution on parallel computing machines is naturally represented using MeshSets, with one type of MeshSet defining the spatial domain and another type representing communication interfaces with other processors; and relations between levels in a multi-level mesh can be stored using MeshSets.

The applications of MeshSets described above rely heavily on conventions defined for various applications. The types of and relations between MeshSets are likely to change, depending on the application, and therefore are easier to describe and change using conventions rather than binding those conventions into the functional interface of MDB. We note that changes to more common types of MeshSets (e.g. geometric topology) should follow standard conventions, in both the pre-processing and analysis communities. In practice, we think this will be quite feasible.

Conclusions

We believe that, though there are costs associated with it, a formal mesh interface such as the TSTT interface is crucial to the continued development of meshing related software at Sandia National Laboratories. The ability to update and refine the mesh data implementation independent of the calling code is in itself sufficient benefit to justify using the interface. Reusability of the code in multiple codes is also a great benefit. We currently plan to use MDB in at least three codes at Sandia in the immediate future.

References

[1] T. D. Blacker et al., 'CUBIT mesh generation environment, Vol. 1: User's manual', SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994, http://endo.sandia.gov/cubit/release/doc-public/Cubit_UG-4.0.pdf

[2] OVERTURE: Object-Oriented Tools for Solving PDEs in Complex Geometries, http://www.llnl.gov/CASC/Overture/.

[3] Scientific Computational Research Center, Rensselaer Plytechnic Institute, http://www.scorec.rpi.edu.

[4] The Terascale Simulation Tools and Technology (TSTT) Center, http://www.tstt-scidac.org/.

[5] SCIDAC: Scientific Discovery Through Advanced Computing, http://www.csm.ornl.gov/scidac/.

[6] Randall M. Summers, James S. Peery, Mike W. Wong, Eugene S. Hertel, Tim G. Trucano, and Lalit C. Chhabildas, "Recent Progress in ALEGRA Development and Application to Ballistic Impacts ", International Journal of Impact Engineering, pp. 779-788 (1997).

[7] SIERRA: Common Code Architecture Enabling Advanced Solution Modules, http://www.sandia.gov/ASCI/apps/SIERRA.html

[8] Larry A. Schoof, Victor R. Yarberry, "EXODUS II: A Finite Element Data Model", SAND92-2137, Sandia National Laboratories, Albuquerque, NM, September 1994, http://endo.sandia.gov/SEACAS/Documentation/exodusII.pdf.

[9] The Verde (Verification of Discrete Elements) tool, http://endo.sandia.gov/cubit/verde_release_2.5b.txt.

[10] Timothy J. Tautges, "CGM: a Geometry Interface for Mesh Generation, Analysis and Other Applications", Engineering with Computers, 17:299-314 (2001).

[11] The GOMA computer code, http://www.esc.sandia.gov/goma/home.html.