Metadata models for ad hoc queries on terabyte-scale scientific simulations

Byung S. Lee, Robert R. Snapp
Department of Computer Science
University of Vermont
Burlington, Vermont, U.S.A.
{bslee,snapp}@cs.uvm.edu

Ron Musick*, Terence Critchlow
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California, U.S.A.
musick@ikuni.com, critchlow@llnl.gov

Abstract

We present our approach to enabling approximate ad hoc queries on terabyte-scale mesh data generated from large scientific simulations through the extension and integration of database, statistical, and data mining techniques. There are several significant barriers to overcome in achieving this objective. First, large-scale simulation data is already at the multi-terabyte scale and growing quickly, thus rendering traditional forms of interactive data exploration and query processing untenable. Second, a priori knowledge of user queries is not available, making it impossible to tune special-purpose solutions. Third, the data has spatial and temporal aspects, as well as arbitrarily high dimensionality, which exacerbates the task of finding compact, accurate, and easy-to-compute data models.

Our approach is to preprocess the mesh data to generate highly compressed, lossy models that are used in lieu of the original data to answer users' queries. This approach leads to interesting challenges. The model (equivalently, the content-oriented metadata) being generated must be smaller than the original data by at least an order of magnitude. Second, the metadata representation must contain enough information to support a broad class of queries. Finally, the accuracy and speed of the queries must be within the tolerances required by users. In this paper we give an overview of ongoing development efforts with an emphasis on extracting metadata and using it in query processing.

Keywords: scientific databases, ad hoc queries.

1 Introduction

As computing power increases, the availability and size of simulation data is growing rapidly. Scientific efforts, such as the DOE Accelerated Scientific Computing Initiative (ASCI), are simulating, with ever increasing precision, large-scale physical systems (e.g., hydrodynamical systems). Simulation data has steadily grown from gigabyte to terabyte scales, with today's largest runs approaching 100 terabytes. Simply moving that much data between two systems is a problem, let alone attempting to analyze and interpret it using conventional visualization tools and database methods. Current approaches require scientists to manually direct the search for anomalies, underlying patterns, and events of interest by *visually* identifying regions while browsing through various representations of the mesh. This becomes infeasible when a single simulation can produce a multi-terabyte dataset with thousands of time steps and billions of zones. It is imperative to develop an interactive *ad hoc query mechanism* that allows scientists to locate regions of interest by specifying them declaratively. Realizing this capability over terabytes of computational data is beyond the reach of current technology.

Visualization tools [6, 7] available to date provide a few fixed forms of primitive query operations such as finding points, and displaying isosurfaces, orthogonal slices, and vector fields. For better efficiency, the data is stored as collections of binary files and accessed with special-purpose engines that provide no support for ad hoc queries. Storing the original terabyte-scale data instead in a database management system (DBMS) may rectify this problem, but such an approach carries a high cost [2]. For example, the data throughput requirements (in terms of megabytes/sec) of visualization applications on large mesh data are substantially higher than what current relational DBMSs can support, so this leads to a significant reduction in performance. Besides, storing the original data in a database multiplies disk storage requirements, thus further aggravating query performance. In short, this straightforward approach is simply not practical.

*Currently with iKuni, Inc., Palo Alto, California, U.S.A.

We approach this problem by building compact, approximate, multi-resolution models of the mesh data and then using the models to support high-fidelity ad hoc queries. The models, along with the access methods needed for using the models, are constructed in a preprocessing phase. Statistical and mathematical data analysis and compression methods such as spline-based fitting [16, 19], wavelets [20, 21], and clustering [8, 9] are all suitable in the preprocessing phase.

Our goal is to produce a suite of *metadata* (including both the models and the access methods pertinent to them) that are at least an order of magnitude smaller than the original mesh data, thus making the use of DBMS technology feasible. The metadata is stored and accessed through an object-relational DBMS (ORDBMS), and the ORDBMS's query engine plays an important role in helping support ad hoc queries. ORDBMSs have the potential to provide high data throughput rates, and allow great flexibility in where and how the metadata is stored and accessed. The metadata-based query processing engine is interfaced to existing mesh visualization packages so that query results can be displayed graphically to the user. Figure 2 in Section 4.2 describes the architecture of the system that realizes these ideas.

There are many challenges to overcome for this approach to work, some of which are listed below. Resolving them involves investigating and making complex tradeoffs between preprocessing time, compression level, query speed, query accuracy, and the range of queries the metadata supports.

- Scalable solutions: all aspects of the preprocessing phase must be scalable to terabyte mesh data, and the algorithms must be amenable to efficient parallelization on machines with hundreds to thousands of processors.

- Complete models: little is known about what makes an effective model of mesh data that is capable of supporting arbitrarily complicated ad hoc queries. There are many alternatives which have varying degrees of accuracy and descriptive power. For example, if the metadata supplied is based on averages of variable values over all time steps, answering a rate-of-change query becomes impossible. The models must be compact, yet contain enough information to answer a wide range of possible queries.

- Compact, efficient, multi-resolution metadata: we allow the scientists to trade query *response time* for *accuracy*, allowing interactive ad hoc queries at one extreme, and slower, more accurate responses at the other. This places several hard constraints on the metadata. It must provide a highly compressed multi-resolution model of the mesh data without sacrificing efficient operations for compressibility. Different models not only lead to a variety of query processing mechanisms and speeds but also potentially require different mechanisms for providing multi-resolution support.

- Accurate models and high dimensionality: including spatial and temporal coordinates (e.g., x, y, z, t) along with the field variables computed by the simulation, mesh dimensionalities typically range between 10 and 100. High dimensionality makes the problem of building accurate models much more difficult, adversely impacting the time to solution, compressibility, and modeling error.

- Quantitative error models: a framework for quantifying expected query errors is needed if users are to be able to trade accuracy for query performance. The challenge is to design a framework that can accurately measure the expected error incurred by a query over multiple models both within and across partitions.

Currently, an effort is under way to build a series of prototypes that will eventually be applied to terabyte-scale mesh data. The prototypes are extensible to allow for the addition of new preprocessing methods as well as their associated metadata, access methods, and rules for choosing access methods for a given query type. The rules mentioned above also help determine which level in the multi-resolution representation should be used in order to generate an answer within a specified error tolerance.

The prototype is usable not only with simulation data but also for experimental data with grid structures in it (for example, sensor data aligned in a grid), and GIS systems in general. We also see a great potential for enhancing the scalability of business intelligence and data warehousing applications, which share the need to process large-scale multi-dimensional data.

In the rest of the paper, we first provide a simplified model of mesh data in Section 2. Then, in Section 3 and Section 4 we sketch the ongoing implementation of our metadata approach to ad hoc mesh query processing. Related work is described in Section 5, and conclusion follows in Section 6.

2 Simulation mesh data model

Simulation mesh data is generated through numerical calculations, typically in a sequence of time steps. The data point grid may be either regular or irregular, time-variant or time-invariant, and sparse or dense [1]. A mesh data model comprises the topology of data points as well as the spatiotemporal configuration of them. In this paper we focus on the spatiotemporal aspects only. (Readers are referred to [2] for a more detailed discussion of mesh data.) The data structure and query semantics of mesh data overlap those of spatial data significantly. Nonetheless, there exist distinctions in the following aspects: (1) mesh data points form an explicit grid structure; (2) simulation time step, if treated separately from spatial variables, implies the semantics of time series data; (3) generated simulation data is read-only.

Let us define mesh data as a discrete representation of continuous data, which can be defined as an ordered set of tuples as follows. (Mesh data represented in this manner is called a *point mesh* [3], which is just a collection of data points with no topological connection among them.)

$$\{< t, x_1, x_2, \cdots, x_n, v_1, v_2, \cdots, v_m >\} \qquad (1)$$

where t denotes a *temporal variable* defining a time step, $x_i, i \in [1..n]$, denotes a *spatial variable* defining the geometrical coordinates in an n-dimensional space, and $v_j, j \in [1..m]$, denotes a *field variable* defined at each node (positioned at $< t, x_1, x_2, \cdots, x_n >$) or each zone in the mesh at time step t. A zone in a *regular* mesh is an n-dimensional cubic bounded by the surrounding 2^n mesh nodes whereas a zone in an *irregular* mesh is surrounded by an indeterminate number of nodes.

3 Metadata extraction

In this section we first describe the configuration of metadata–what makes up the elements and how they are organized. Then we introduce our approach to creating it, for which we first elaborate on the spline-based modeling and then skim other alternative modeling methods.

Recall that the goal of preprocessing the mesh data is to generate a representation of the model that is much smaller and yet can be used to support approximate ad hoc queries. Here we adopt a flexible approach that allows the metadata to be constructed in a piecewise fashion. First the entire mesh is decomposed into an appropriate partition, then the mesh data within each partition is concisely approximated using an appropriate parametric model (e.g., splines, wavelets, or clusters). Algorithms to generate this metadata typically iterate through two phases: *mesh partitioning* and *partition characterization*. Iteration between these phases takes place to revise the actual partitioning based on a similarity (or error) metric that measures the difference between the characterization of the data in a partition and the actual mesh data itself.

3.1 Metadata content and organization

We organize the metadata in two levels. The lower level contains two elements. The first element is a *multi-resolution model* of mesh data, modeled per partition of mesh data, as well as a collection of *summary* information that is generated as the result of preprocessing. For example, the summary information includes, per variable, information on min, max, mean, median, standard deviation, and the first several moments for each variable. The actual models (described in more depth below) vary from cluster prototypes, to regression equations, to matrixes of equations, and so on. The second element of metadata is a set of *indexes* which define the structure that must be traversed to reach the first element data. More specifically, each node of an index contains the summary information and model information of the corresponding mesh partition. The summary information contains pre-calculated aggregate values at different levels of grouping, analogous to materialized views [25] in data warehousing. The model information contains the preprocessing method used to model that partition, the generated model data, and the modeling error. Each index node also contains the bounding box information, i.e., the size and location of the mesh partition covered by the node. Additionally, an internal node in the index contains the links for binding all child nodes in a manner similar to that of a multi-dimensional spatial access method. Figure 1(a) illustrates the structure of a two-dimensional index tree, where each index node is depicted by a rectangle indicating the corresponding quadtree partition of mesh data. The structure of an index node is shown in Figure 1(b).

The higher-level metadata contains information that is independent of the particulars of different preprocessing approaches and is used to help organize the

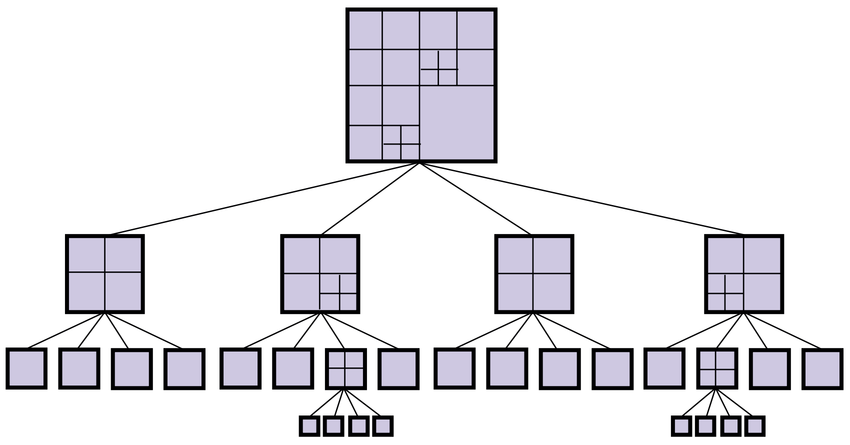


Figure 1: An example of index tree structure.

query processor. More specifically, it contains the following three elements. The first is information about the most efficient access paths for queries of a given type (see Table 2 in Section 4.2) as well as estimators of the query response times. The second element is a mapping from a preprocessing method to a preferred program object (called a *metadata processor*). A program object "knows" how to reconstruct approximate mesh data from the model that is generated in the preprocessing stage. The third element is semantic information such as locations of empty regions on the mesh and dependency information between correlated field variables.

3.2 Spline-based modeling

The spline-based fitting is one approach to generating an approximate replica of the data. Splines are essentially a set of basis functions which can be found at the root of many wavelet basis functions and regression approximations. Splines serve as a useful polynomial model of data that originates from the smooth dynamical systems often encountered in scientific applications. Polynomials are computationally efficient: they are easy to store, manipulate, and evaluate. Derivatives and integrals, which may need to be evaluated in order to answer certain queries, can be obtained for splines algebraically. Similarly, the roots (or zeros) of polynomials are readily obtained by standard numerical methods. The use of splines to model piecewise-continuous phenomena is supported by a large body of theory [17]. For instance, every function that is continuous over a finite interval can be approximated to arbitrary accuracy using splines of fixed order, provided that a sufficient number of knots (i.e., polynomial segments) is allowed. Furthermore, rates of convergence of spline approximations, and their derivatives for large classes of continuous functions are also well known.

There are several different splines, but *B-splines* [19] are used most popularly. Of particular interest is that they have been used as a method for compressing large observational data sets [18]. Multivariate surfaces can be defined in n dimensions in terms of tensor-product splines [16], where the vector-valued control points are estimated by minimizing the sum of squared errors between the tensor-product spline and the mesh data. For example, consider a data set defined on an n-dimensional rectangular mesh $\vec{V}(x_1, \ldots, x_n)$ where the coordinate variable $x_j, j \in [1..n]$, independently assumes ρ_j values $\{x_{j,1}, \ldots, x_{j,\rho_j}\}$. For $i = 1, \ldots, n$, let k_i denote the number of knots along the x_i coordinate axis, and let$\{B_{i,1}(u), \ldots, B_{i,k_i}(u)\}$ denote the corresponding set of B-splines. A tensor-product spline can then be defined as:

$$\vec{F}(x_1, \ldots, x_n) = \sum_{i_1=1}^{k_1} \cdots \sum_{i_n=1}^{k_n} \vec{p}_{i_1 \ldots i_n} B_{1,i_1}(x_1) \cdots B_{n,i_n}(x_n) \tag{2}$$

where the control points $\vec{p}_{i_1 \ldots i_n}$ specify the shape of the generated surface. The values of the control points are chosen to minimize the approximation error

$$\sum_{j_1=1}^{\rho_1} \cdots \sum_{j_n=1}^{\rho_n} ||\vec{F}(x_{1,j_1}, \ldots, x_{n,j_n}) - \vec{V}(x_{1,j_1}, \ldots, x_{n,j_n})||^2 \tag{3}$$

A variety of statistical techniques (i.e., regression) have been studied, including efficient implementation strategies and methods for adaptively modifying the number and location of knots.

Our spline-based modeling algorithm has two phases – *partitioned spline-based fitting* and *index tree construction* – that are interleaved in an actual implementation. The former phase has two steps: *partition* and *fit*. In the *partition* step we multi-dimensionally bisect the mesh data, and in the *fit* step we perform spline-based fitting against the data in each partition. For mesh data in an n-dimensional space, each partition is 2^n-th the size of its parent partition. (This makes an octree structure if $n = 3$.) If the fitting error of a partition exceeds the specified maximum allowed error, we partition it further into subpartitions. This partition-and-fit is repeated recursively until every partition of data is fitted within the maximum allowed error. In the index tree construction phase, we create one node for each partition. A node contains, as the model data, the control points (i.e., $\{\vec{p}_{i_1 \cdots i_n}\}$) and knot locations of the B-splines used in the corresponding partition. In addition, it contains the summary information and bounding box information described in Section 3.1.

The granularity of the fitted data can be a field variable or a vector of field variables (e.g, $\vec{v} \equiv < v_1, v_2, v_3, \ldots >$). A key question is which set of indexes to create over the vector of variables. The most elementary approach is to create one index per field variable. This supports a wide range of queries, but better compression and error modeling is likely to result from modeling highly correlated variables together. Since the potential number of unique multi-variable combinations is exponential with the number of variables, we need an organized way to limit the number of composite indexes in consideration. To this end, we either use probabilistic reasoning such as the qualitative structure of a belief net [24] learned from sample points or in the tuning phase wait and see which queries are common, and then jointly index the highly correlated variables that tend to appear together.

3.3 Alternative modeling methods

Alternative characterizations: Tradeoffs exist between different approaches to modeling the data in each partition. All these methods share the theme of providing a compressed, approximate representation of the original data. Note that in most cases the characterization of the data is tied to the error metric used in the partitioning phase.

The simplest model we consider is a *prototype object* that depicts the centroid of each partition. This model consists of, for example, the mean and several moments of each field variable in the cluster. Since in fact this data is already a part of our summary metadata, representing such cluster prototypes requires no additional effort. Of course, the model it provides is also the least informative, and so likely requires very small clusters before being able to satisfy the error constraints. This model can be made more complex and informative by representing the prototype components with mixtures of *distributions*. Data permitting, we can use the data to form sufficient statistics for a mixture of beta or Gaussian distributions per variable.

An alternative characterization method that is more complex and informative than the spline-based fitting (described in Section 3.2) is the use of *wavelets*. Data in each partition can be modeled with the wavelet representation resulting from lossy wavelet-based compression of the data. A wavelet representation can be viewed as a matrix of locally dependent linear equations composed of different basis functions. Wavelet-based methods for compressed multi-resolution representations of mesh data have been successfully used for large-scale visualization problems [22].

Among the tradeoffs between the alternative methods is how the cost of partitioning the data (as well as the size and number of partitions) varies with the complexity and accuracy of the resulting model. The issue of tuning is also important here. Since we are dealing with simulation data, it is likely that several variables in the mesh are functionally related. If we know a priori that such relationships exist, and the scientists can pass those relationships on, those functions can be represented explicitly in the metadata, thus in effect reducing the dimensionality of the overall problem.

Alternative partition spaces: The partitioning we have described in Section 3.2 has been along spatiotemporal axes. There are many other ways to break up mesh data into smaller collections of objects. The mesh data can be mapped into a long time series aligned by the simulation time step, which could then be compressed with techniques like *principal components analysis* [5]. This modification would require non-trivial changes to the approach we described for indexing and querying the multi-resolution metadata.

A different approach is to treat the spatial and temporal variables no differently than the field variables. In this case, a mesh is converted to a large set of *hyper-dimensional points*, or *feature vectors*, identical to the typical datasets found in the data mining community. We are exploring clustering over a sample of points in this space, iteratively refining clusters as the rest of the data is read in. The advantage of this approach is that clusters may be more compact in this space, leading to better compression rates. The disadvantage is that we need to associate the hyper-dimensional clusters to spatiotemporal regions in order to answer region-dependent queries. Vector quantization combined with bit map indexing is likely to make this feasible.

Query forms relating to the target and predicated objects:

SR	Find summaries of variables given the specification of mesh regions.
RS	Find a mesh region given the specification of mesh subregions.
SS	Find summaries of variables given the specification of mesh subregions

Query conditions relating to the number of time steps:

1T	One time step.
NT	N serial time steps, where $N > 1$.

Query conditions relating to the number of predicate variables (M):

NV	No variable (i.e., $M = 0$)
UV	Univariate (i.e., $M = 1$)
MV	Multivariate (i.e., $M > 1$)

Table 1: Classification of query types.

4 Query support using metadata

4.1 Query classification

In order to rationally choose between different types of compact models, it is important to understand the types of queries that can be addressed to mesh data. We categorize queries into the orthogonal types shown in Table 1 to better understand how the metadata we produce and the algorithms for producing the metadata impact the types of ad hoc queries we can ask.

First, the query form is determined by three possible combinations of the target (i.e., retrieved) objects and the predicated objects: SR, RS, and SS. The first letter, either 'S' or 'R,' distinguishes the target objects into *summaries* and *regions*, where a summary refers to variables, functions of them, or aggregations of them, and a region refers to mesh data points that are connected together. The second letter, either 'R' or 'S,' distinguishes the predicated object into *regions* and *subregions*, where regions are specified with predicates solely on only the spatiotemporal variables and subregions are further specified with predicates on field variables as well. Note that a query form 'RR' is pointless because of its tautological semantics of "find the regions of mesh data in the specified regions." Second, the query condition is determined by the number of specified time steps (i.e., 1T and NT) in one way and by the number of predicate variables (i.e., NV, UV, MV) in the other way. Combining the three categories gives us eighteen query types resulting from the following product: { SR, RS, SS } × { 1T, NT } × { NV, UV, MV }. The rationale behind these classifications will become clear as we discuss query processing in Section 4.2.

Example queries are shown below using SQL-like syntax on the mesh data of Equation 1 along with an explanation of what is returned and how the query is classified. We assume the following in these examples: the mesh data filename is "Meshdata"; the system supports a data type "region" and its internal implementation such as a region identity; the system uses "timestep" as a reserved keyword for the simulation time step, which is a temporal variable; x_1, x_2, and x_3 are spatial variables; v_1, v_2 and v_3 are field variables.

Q1 select avg(v_1), avg(v_2) from Meshdata where timestep = 8 and x_1 between 200.0 and 300.0 and x_2 between 1000.0 and 2000.0 and x_3 between 500.0 and 800.0; ⇒ returns the summary (i.e., average) of the fields v_1 and v_2 in the mesh region bounded by the predicates on x_1, x_2, and x_3 at time step 8. This query runs at one time step, therefore 1T, and involves multiple (one temporal, three spatial, and zero field) variables, therefore MV with $M = 4$. [SR.1T.MV]

Q2 select region from Meshdata where timestep between 4 and 1000 and $x_2 > 1000.0$ and (x_3 between 500.0 and 800.0 or $x_1 < 200.0$) and v_1 between 0.0 and 100.0 and v_2 between 50.0 and 150.0 and v_3 between 100.0 and 250.0; ⇒ returns the subregions that are confined within the regions defined by the predicates on timestep, x_1, x_2, and x_3 and further bounded to subregions where the predicates on v_1, v_2, and v_3 are satisfied. This query runs in 995 time steps, therefore NT, and involves multiple (one temporal, three spatial, and three field) variables, therefore MV with $M = 7$. [RS.NT.MV]

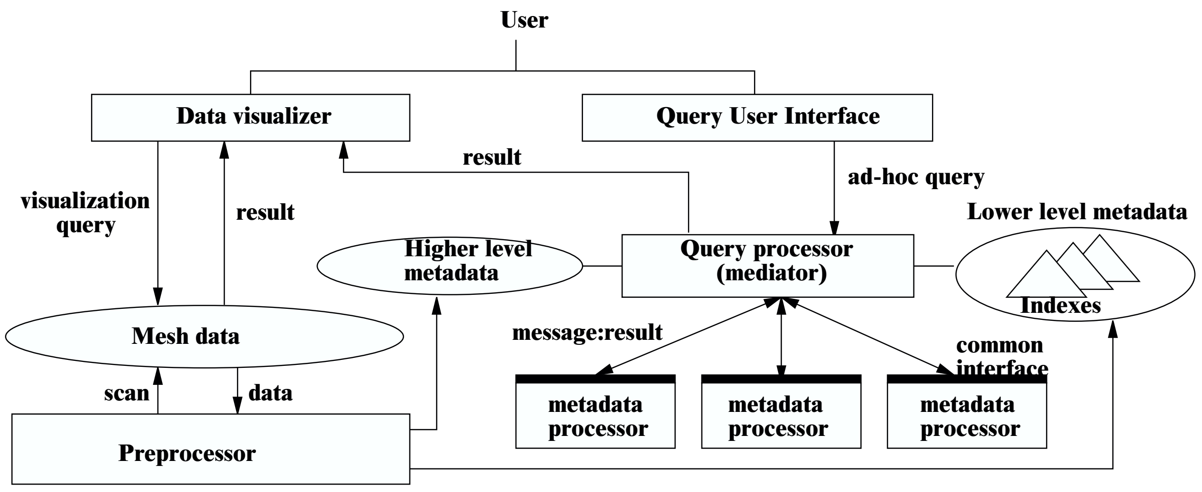


Figure 2: Architecture of the prototype.

Q3 select min(v_1), avg(v_1), max(v_1) from Meshdata where timestep = 13 and x_1 between 200.0 and 300.0 and $x_2 > 1000.0$ and v_2 between 123.0 and 234.0; $\Rightarrow$ returns the three summaries of v_1 in the mesh subregions confined within the regions defined by the predicates on x_1 and x_2 at time step 13 and further bounded to subregions where the predicate on v_2 is satisfied. This query runs at one time step, therefore 1T, and involves multiple (one temporal, two spatial, and one field) variables, therefore MV with $M = 4$. [SS.1T.MV]

4.2 Query processing

Architecture: Figure 2 shows the architecture of the prototype we are building incrementally. In the metadata extraction stage, the preprocessor generates a collection of indexes built upon the mesh data, while using possibly different methods on different partitions of the same mesh data. The preprocessor generates other metadata elements as well, as described in Section 3.1, and stores them in the two layers of metadata. Remember that the higher level metadata contains information for mapping from preprocessing methods to metadata processors.

In the query processing stage, a scientist uses the query user interface to compose an ad hoc query. The query processor then parses the query and categorizes it into one of the pre-defined query types described in Table 1, and determines which indexes to use based on the rules described in Table 2. The query processor then searches the chosen indexes for the nodes that are needed to satisfy query predicates within the limit of a user-specified modeling error. Then, if the query is not answered completely from the pre-calculated summaries stored in the index nodes, it identifies the preprocessing methods from the accessed nodes. It then looks up the mapping information in the high level metadata and decides which metadata processors to invoke for the identified preprocessing methods. Each metadata processor knows how to regenerate the mesh data (approximately) and calculate summaries. Metadata processors return the results (i.e., regions or summaries, depending on the query type) to the query processor, which then integrate the results and sends the integrated data to the visualization engine on the desktop for display.

All metadata processors have a common interface (i.e., public methods) that defines a metadata model. We store the metadata within an object-relational DBMS if performance requirements can be met, otherwise outside the DBMS storage manager and link to the data using the ORDBMS's mechanisms for tying external data into the DBMS's query engine.

Procedure: Queries are processed using the index trees built in the preprocessing stage as follows: choose one of indexes that is optimum for a given selection condition, search the tree down to find a set of nodes that satisfy the query condition and meet the error constraint, and regenerate the mesh data of the corresponding partitions. Table 2 summarizes our approach to managing different types of queries given a rich set of indexes from which to potentially answer the query. Specifically, the table lists the appropriate indexes to be searched by a query processor given

Query forms relating to the target and predicated objects:

SR	Search indexes available on the target field variables for the nodes encompassing the mesh regions specified by the predicate on spatial variables. For each found node, retrieve the target summaries if pre-calculated, otherwise calculate them from the model (e.g., coefficients) stored in the node.
RS	Separate the predicate variables into spatial variables and field variables. Search indexes available on the field variables for the nodes whose contained min-max range falls within the specified range of values. Consolidate all the mesh regions covered by the found nodes and then intersect the resulting regions with those bounded by the predicates on the spatial variables.
SS	Given the predicate variables, find mesh regions in the same manner as in RS and, given the resulting mesh regions, find the target summaries in the same manner as in SR. (Note that SS is equivalent to RS followed by SR.)

Query conditions relating to the number of time steps:

1T	Prefer separate indexes for each time step.
NT	Prefer one index across time steps.

Query conditions relating to the number of predicate variables (M):

NV	Use no index, in effect performing an exhaustive scan of the entire mesh data.
UV	Use simple indexes on the predicate variables.
MV	Prefer composite indexes on the predicate variables and, if none exists, simple indexes on individual variables.

Table 2: Rules for using indexes for query classes shown in Table 1.

associated query classes in each category.

A few points are worth mentioning regarding indexes in Table 2. First, an "index" denotes a *database* index that stores data (e.g., coefficients) generated from preprocessing mesh data and is accessed (i.e., searched) by a query processor. Second, the same index can be used for searching based on either variable values or mesh regions, or both. This *dual usage* is possible because an index node stores the size and location of the mesh regions the node covers as well as the min–max range of the indexed variable or vector. Third, we always prefer to use a *composite index* (i.e., an index created on a vector) whenever applicable, for efficiency reasons. Fourth, we assume that indexes are created on the *values* of a variable, not functions of values.

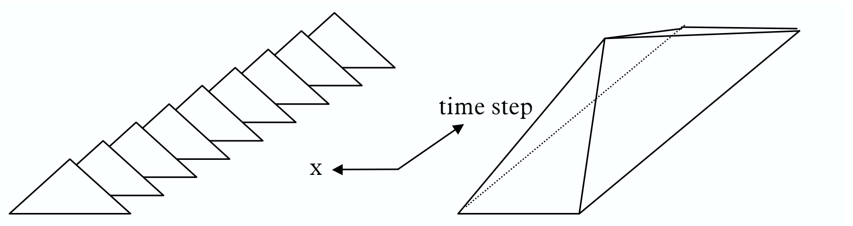
If there are highly *correlated* variables v_i and v_j, an index may only need to be created on either variable, not both, and this fact is recorded as part of the higher level metadata. If the correlation is quantifiable, that is, can be modeled as an invertible conversion function, then the query processor uses an index on one variable even if the other variable is predicated in the query condition. For example, if $v_i = 2 \times v_j + 1$ and an index exists on v_i, then a query selection condition $v_j > 4$ can be processed using an index on v_i for a condition $v_i > 1.5$.

In the example below, we suggest the most preferred index and then sketch the query processing using the index for each of the query examples shown in Section 4.1.

Q1 The best is to have a composite index on v_1 and v_2 at time step 8, with pre-calculated averages of v_1 and v_2 stored in the index nodes. Given all the found index nodes, which overlap the spatial bounding box predicated on x_1, x_2, and x_3, calculate avg(v_1) and avg(v_2) from the stored value assuming the uniform distribution of the values of v_1 and v_2. Knowing the actual distribution certainly contributes to the accuracy but to the computational cost as well. If there is no pre-calculated avg(v_1) or avg(v_2), calculate it from the model stored in the index node.

Q2 The best is a composite index on v_1, v_2, and v_3 across time steps including 4 to 1000. As the index search returns a set of nodes based on the predicates on v_1, v_2, and v_3, subregions of the partitions corresponding to the nodes are selected from the spatiotemporal bounding boxes specified with timestep, x_1, x_2, and x_3 by applying the field predicates on v_1, v_2, and v_3. Applying a field predicate requires using a numeric solver.

Q3 The best is a simple index on v_2 at time step 13, with pre-calculated min, avg, and max of v_1 stored in each index node. As in Q2, select mesh subregions based on the spatial bounding boxes specified with x_1 and x_2 at time step 13 and the field predicate on v_2. Then, as in Q1, calculate the summaries of v_1 in the selected subregions



(a) one composite index per time step. (b) one simple index across time steps.

Figure 3: An example of 1T composite indexes and an NT simple index.

out of the stored values under the assumption of a uniform distribution.

In the unavailability of the desired index, we should compare different options to make the next best choice. We consider two cases of comparisons here. First is between a *composite* index *without* pre-calculated summaries and a *simple* index *with* pre-calculated summaries. The former leads to accessing fewer index nodes but doing more computations (for calculating the summaries) than the latter. One heuristic rule in this case is to favor the former over the latter assuming that calculating summaries in main memory is cheaper than fetching index nodes from the disk. Under this rule, the second best in Q1, for instance, is a composite index on v_1 and v_2 without the pre-calculated averages, and the next best is a simple index on either v_1 or v_2 with the pre-calculated averages. (Making a systematic choice between v_1 and v_2 requires knowing the distribution of their values relative to the predicates on them, similarly to a conventional query optimization.)

The second case is between a *1T composite* index, created per time step, and an *NT simple* index, created across multiple time steps. (Figure 3 illustrates this with a two-dimensional index.) The former leads to searching more indexes (i.e., as many as the number of queried time steps) but accessing fewer nodes per index than the latter. The number of 1T simple indexes searched for a given query is bounded by the number of actually indexed variables. The choice of an index in this case depends on the expected number of accessed index nodes, which increases logarithmically with the number of queried time steps for the NT simple indexes whereas linearly for the 1T composite index. For instance, the second best in Q2 may well be three simple indexes on each of v_1, v_2, and v_3 created across time steps including 1 to 1000.

5 Related work

To the best of our knowledge, there is no existing system that supports ad hoc queries on large-scale mesh data. There is an extensive body of literature from the visualization community about the visual exploration of computational data. Many current efforts are, for example, exploring schemes to generate multi-resolution representations of large surfaces [22, 23] to characterize mesh data of varying topology. There is an essential distinction, however, between typical visualization efforts and the goals outlined in this paper. Visualization efforts are not aimed at supporting ad hoc queries, but rather at supporting a much more restrictive set of pre-defined visualization "queries". While many of the underlying algorithms for modeling the data are useful in either setting, this difference in goals results in differences in approaches that range from the initial partitioning of the data to its final characterization.

The STING project as described by Wang et. al. in [4] is a closer fit to the challenges outlined above. STING is described as a spatial data mining approach based on a statistical information grid. STING uniformly partitions the data into rectangular cells, builds a hierarchy on top of them by integrating four cells into one at the higher level of the hierarchy, and stores some pre-calculated summary information (e.g., min, max, mean) in each cell. In addition, STING supports two types of region queries: one finds regions given conditions on variables in the region, the other describes the data in a region. Our query classification scheme (in Section 4.1) extends this into a multidimensional classification, while our mesh partitioning and characterization approaches focus on satisfying more stringent compression, scalability, and usability metrics.

The support for region queries is closely associated with partitioning mesh data. STING builds uniform partitions of data and characterizes the content of each partition. There are a multitude of other approaches that can be applied to the problem of partitioning data into smaller, more manageable pieces. Graph partitioning algorithms such as METIS [11] and a parallelized version parMETIS [12] are commonly used for domain decomposition problems in scientific computing. While interesting, these algorithms are optimized for minimizing the number of edges that are cut in a partitioning, which is hard to map to more generalized error (or similarity) metrics required for the partitioning we need to carry out. Of more immediate applicability are clustering approaches. Clusters are formed by maximizing inter-cluster similarity and intra-cluster differences between groups of objects, as measured by a similarity metric. Clustering approaches can be grouped into two categories, agglomerative and divisive. Agglomerative methods such as Birch [10] and CLARANS [13] are based on k-means, which builds clusters by merging one object or cluster at a time. Divisive approaches such as CLIQUE [15] and the adaptive binning used in MAFIA [14] build partitions by starting with the full set of objects, then dividing or adjusting the partitions recursively. In both cases, decisions on modifying cluster membership are made based on the change that improves the similarity metric the most.

6 Conclusion

We have described an ongoing effort for providing approximate answers to ad hoc queries on mesh data generated through numerical simulations. The approach is to preprocess the mesh data to generate metadata and then use the metadata to answer users' queries approximately. Based on a simplified view of mesh data as a collection of spatiotemporal data points, we have described the configuration and extraction mechanism of metadata. Specifically, we have given an overview of multi-resolution index construction that uses mesh partitioning and partition characterization recursively, presented the spline-based fitting as one viable modeling technique, and sketched other alternative modeling methods. Then, we have classified query types based on anticipated uses of the metadata, and presented query processing architecture and procedure associated with the query types. The architecture has been designed to incorporate multiple modeling techniques as the system evolves. We have also provided an example for making a logical connection between the metadata, query types, and query processing.

Acknowledgment

For LLNL authors, work was performed under the auspices of the U.S. DOE by LLNL under contract No. W-7405-ENG-48. For UVM authors, research was partially supported by the University of Vermont UCRS Faculty Research Grant No. PSCI00-1.

References

[1] A. Shoshani, F. Olken, and H. Wong, "Characteristics of Scientific Databases," *Proc. Intl. Conf. on Very Large Databases*, 1984, pp. 147-160.

[2] R. Musick and T. Critchlow, "Practical Lessons in Supporting Large-Scale Computational Science", *ACM SIGMOD Record*, Vol. 28, No. 4, December 1999, pp. 49-57.

[3] http://www.llnl.gov/meshtv/manuals.html, Silo Documentation Version 4.0, Lawrence Livermore National Laboratory.

[4] W. Wang, J. Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," *Proc. Intl. Conf. on Very Large Data Bases*, 1997, pp. 86-195.

[5] J. Jackson, *A User's Guide to Principal Components*, John Wiley, 1991.

[6] http://www.llnl.gov/bdiv/meshtv/, MeshTV: Scientific Visualization and Graphical Analysis Software.

[7] http://www.atmos.uiuc.edu/envision/envision.html, ENVISION: an Interactive System for the Management and Visualization of Large Geophysical Data Sets.

[8] J.A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, 1975.

[9] J. Zupan, *Clustering of Large Data Sets*, Johns Wiley & Sons, 1982.

[10] Tian Zhang, Raghu Ramakrishnan,and Miron Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *Proc. SIGMOD Intl. Conf. on Management of Data*, 1996, pp. 103-114.

[11] G. Karypis and V. Kumar, *Multilevel Algorithms for Multi-Constraint Graph Partitioning*, Technical Report 98-019, University of Minnesota, 1998.

[12] G. Karypis and V. Kumar, "A Coarse-Grain Parallel Formulation of a Multilevel k-way Graph Partitioning Algorithm," *Proc. SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[13] R. T. Ng, and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," *Proc. Intl. Conf. on Very Large Databases*, 1994, pp. 144-155.

[14] S. Goil, Harsha, and A. Choudhary, *MAFIA: Efficient and Scalable Subspace Clustering for Very Large Datasets*, Technical Report, Northwestern University, 1999.

[15] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1998, pp. 94-105.

[16] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978.

[17] L. Schumaker, *Spline Functions: Basic Theory*, Wiley: New York, 1981.

[18] G. T. Anthony and M. G. Cox, "The Fitting of Extremely Large Data Sets by Bivariate Splines," in J. C. Mason and M. G. Cox, ed. *Algorithms for Approximation*, Clarendon Press, Oxford, 1987, pp. 5-20.

[19] P. Dierckx, *Curve and Surface Fitting With Splines* (Monographs on Numerical Analysis), Oxford University Press, 1993.

[20] A. Antoniadis and G. Oppenheim (ed.), *Wavelets and Statistics*, Springer-Verlag, 1995.

[21] W. Härdle, et al., *Wavelets, Approximation, and Statistical Applications*, Springer-Verlag, 1998.

[22] M. Bertram, M.A. Duchaineau, B. Hamann, and K.I. Joy, "Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization," *Proc. IEEE Visualization Conference*, 2000, pp. 389-396 & 579.

[23] M. Eck and H. Hoppe, "Automatic Reconstruction of B-spline Surfaces of Arbitrary Topological Type," *Proc. ACM SIGGRAPH*, 1996, pp 325-334.

[24] R. Musick, "Rethinking the Learning of Belief Network Probabilities," *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*, August 1996, pp. 120-125.

[25] I.S. Mumick and A. Gupta (ed.), *Proc. Workshop on Materialized Views: Techniques and Applications* in cooperation with ACM SIGMOD, June 1996.