# A Data Model for Distributed Multiresolution Multisource Scientific Data

Philip J. Rhodes, R. Daniel Bergeron, and Ted M. Sparr

University of New Hampshire, Durham NH 03824

**Summary.** Modern dataset sizes present major obstacles to understanding and interpreting the significant underlying phenomena represented in the data. There is a critical need to support scientists in the process of interactive exploration of these very large data sets. Using multiple resolutions of the data set (multiresolution), the scientist can identify potentially interesting regions with a coarse overview, followed by narrower views at higher resolutions.

Scientific data sets are often multisourcecoming from different sources. Although it may be infeasible to physically combine multiple datasets into a single comprehensive dataset, the scientist would often like to treat them as a single logical entity. This paper describes formal conceptual models of multiresolution and distributed multisource scientific data along with an implementation of our multisource model. Our goal is to allow a scientist to describe a dataset that combines several multisource multiresolution datasets into a single conceptual entity and to provide efficient and transparent access to the data based on functionality defined by the model.

## 1 Introduction

New data gathering and data generation tools have created an explosion of data available to scientists. The existence of such large amounts of data opens up a wide range of opportunities for scientific data exploration. Widespread availability of ever increasing compute power provides some hope that we can realize these opportunities. The size and nature of the data, however, do present substantial obstacles.

Multiresolution data consists of several representations of a dataset at various resolutions, ranging from the original resolution to a very coarse overview. The scientist can use the overview to identify significant regions of the data and then examine such regions in greater detail using one of the finer resolutions.

The problems presented by large data size are especially difficult when a scientist wishes to combine a set of related multisource data sets into a single conceptual entity. In order to access such data efficiently, we need to understand the fundamental nature of the data and capitalize on its inherent structure.

This work is supported by the National Science Foundation under grants IIS-0082577 and IIS-9871859

We have developed a formal data model and an implementation of that model that supports a wide range of scientific datasets with a variety of data organizations [16]. The complete model encompasses hierarchies of multiresolution and adaptive resolution data in a distributed environment.

Our model and our implementation is intended to support a concept analogous to a database view. For example, a scientist ought to be able to define a dataset representing points in a three-dimensional space with 4 data attributes, where one of the attributes is defined (for all points) in a single physical file and the other 3 attributes are partitioned into spatial blocks which are distributed to multiple locations across the web. The composite dataset will never be instantiated as a single object, but should appear to the user as a single unified entity.

We first describe the kind of data we wish to model and then give an overview of our formal data model. We present a brief summary of our implementation of the multisource data model and present a specific example describing how a scientist might define and access a composite multisource data set. We conclude with some observations and directions for future development.

### 1.1 Problem Definition

The research described here focuses on developing database support for this method of scientific research based on interactive exploration of very large distributed multiresolution data. We believe that a major weakness of current scientific database efforts is the lack of a comprehensive model that encapsulates the structure inherent in the data. Such a model should allow a database system to store and access this data efficiently without needing to understand the meaning of the data for the application domain. The most important requirements for a data model for distributed multiresolution data include the following:

- The model must be general-purpose while still able to rigorously encapsulate the most important aspects of the data.
- In addition to describing the multiple resolution levels of a data set, it must be able to describe an *adaptive resolution* level, i.e., a single level of the data set that is itself composed of data that has different resolutions in different regions.
- It must be able to describe both the physical domain in which the scientific phenomenon actually occurs (we call this the *geometry* of the problem) as well as the structure of the data that represents that phenomenon (which we call the *topology* of the data).
- It must incorporate information about how data points in each level of the multiresolution representation relate to data points in the other levels. We have developed notions of *support* and *influence* which encapsulate these relations in an application-independent manner.

- It must allow scientists to assemble data from multiple local or remote sources into a single conceptual entity.

### 1.2 Summary of Research

The major components of our research include:

*MR/AMR Model.* We are developing a data model for hierarchical multiresolution (MR) and adaptive multiresolution (AMR) data representations that supports interactive exploration of scientific data. This includes a model of error and error operations that helps keep the experimenter informed of the quality of data at various resolutions. Also, we characterize the kinds of operations that can be performed on MR/AMR data, especially as they relate to data in a distributed computing environment.

*Geometry and Topology.* Our data model distinguishes between the geometry and topology of a dataset, allowing us to characterize a wide variety of data types. This work should allow us to develop a taxonomy of scientific data that helps exploit regularities in both geometry and topology. We see the topology as a bridge between the scientists geometric data view and the index-oriented view of the underlying database.

*Lattice Model.* The lattice model is a single-level model of data that incorporates our ideas about geometry and topology, and is an important component of the formal model especially for representing adaptive resolution data. Geometry and topology forms the basis of a lattice class hierarchy that can efficiently represent a variety of scientific data.

*Domain Representation.* We are developing an efficient way to represent domains, and especially the extent of subdomains within an enclosing domain. The representation should be space efficient and quick to access. This work is particularly important because we represent certain metadata (e.g., extracted features, classifications) as labeled subdomains.

*Data Storage.* We have developed a model for multisource data and implemented a prototype for evaluation. For array-based data, our approach is spatially coherent, i.e., given a point, we have efficient database access to its geometric neighbors.

*Evaluation.* The model will be evaluated by implementing a prototype and testing its performance with large datasets. Our goal is a system that is flexible and expressive enough to be of real assistance as the scientist works with the data.

## 2 Distributed Multiresolution Scientific Data

We envision the creation of a standard multi-level data hierarchy for large scientific data sets . The original data sets would be stored permanently at some repository site or sites. A preprocessing operation would generate a

complete multiresolution representation with increasingly coarse representations and the associated localized error representations. All components of this data representation would be available for downloading arbitrary subsets at arbitrary resolutions. A scientist is likely to extract some small coarse component of the hierarchy to store at his or her workstation, may access the next several levels on a data server on the local network, and perhaps access the finest resolution representations over the Internet. In addition, the scientist would like to define virtual datasets composed of sections or attributes taken from other datasets.

This model works because only very small subsets of the total data set will need to be accessed at the higher resolutions. Note that we can assume that the original data is essentially read-only, although we certainly need to be able to dynamically update both the lower resolution representations and the metadata associated with the data set.

In order for this multi-level storage representation to be effective, the scientist needs to have a cumulative local error value associated with the lower resolution representations of that data. The error representation must be an integral component of the data exploration process providing the scientist with critical feedback concerning where the current representation is likely to be accurate and where it isnt. Without such information, the scientist could not trust anything but the finest resolution.

## 3 Data Model Foundations

Pfaltz et al. [14] identify the major features of scientific data as large size, complex entities and relationships, and volumetric retrieval. Although such characterization is correct, we need a more rigorous definition if we hope to provide effective database support for scientific data. For our purposes, scientific data is a collection of values that represents some natural phenomenon [7] that is a function over a domain [11] which might be time, space, radio frequency, etc. or some multidimensional combination. The *value space* of the function defined over the domain usually consists of the cartesian product of the value ranges of several data attributes. This is equivalent to saying that any point in $D$ has a number of attributes – the value of the data function at that point.

### 3.1 Dimensional Data

Much scientific data can be meaningfully represented in a continuous n-dimensional data space [2,8]. If a data set consists of *some* attributes that are ordinal, independent, and defined on a continuous value range, the data set contains *dimensional data*, and those attributes are *dimensional*. Each possible combination of dimensions defines a *view* of the data, a notion similar to the view capability found in traditional databases. *Spatial data* is dimensional

data that represents an actual physical space. A data set can be dimensional without being spatial but even non-spatial dimensional data can often be visualized as if it were spatial, since humans find this representation familiar and easy to grasp. It may also be convenient to treat a set of attributes as if they are dimensional attributes even though they may not satisfy all the conditions for dimensional data. For example, we might want to treat a set of attributes as independent for exploration purposes with the goal of either validating or disproving that assumption.

Spatial (dimensional) data is often represented as points defined on a wide variety of regular and irregular grid types [2,4,6,18]. The choice of the grid, usually based on a natural organization of the data, impacts the nature of the representation chosen for the data and the specification of algorithms for analyzing it.

### 3.2 Geometry, Topology, and Neighborhoods

The terminology used in the literature to describe various systems of grids is not standardized. We propose a more comprehensive and consistent framework for describing and defining grids that encompasses most reported grid structures [9,11], including both point and cell data organizations. We separately represent the underlying space in which the grid is defined, which we call the *geometry*, and the point and cell relationships implied by the grid, which we call the *topology*. Thus, the geometry of a data set refers to the space defined by the dimensions; the topology of a data set defines how the points of the grid are connected to each other. A data sets topology is a graph with data points or cells as nodes and arcs between nodes representing a *neighbor* or *adjacency* relationship.

This approach enables database support for application algorithms to process data either geometrically or topologically. In many cases, the topology and/or geometry do not have an explicit representation within the data set because they derive easily from the indexes of an array that stores the data. The array and its index structure compose the *computational space* of a data set. Other more complex geometries and topologies may have a separate representation from their computational spaces.

In some applications the data has no inherent geometry or topology. For example, categorical data is normally not defined in a geometric space and scatter data has no predefined topology. Our data model allows a user to represent and manipulate both kinds of data. However, it may be useful to impose additional structure on the data. For example, we could impose a topology on scatter data either for efficiency of access or to support an alternative conceptual model for the data. Similarly we have shown that topology can be an effective vehicle for imposing a metric space upon categorical data [11,12].

Many scientific applications require selecting the neighborhood of a point [5,12,15]. The neighborhood of a point $p$ consists of points "near" $p$. Nearness

may be defined *geometrically* (e.g., as the set of points within distance $d$ of $p$ in the geometric space) or *topologically* (e.g., as the set of points within $n$ arcs of $p$ in a topological space).

### 3.3 Error

Most scientific data contains some inherent error. This includes measurement error from sampling or computational error from simulation. Furthermore, operations and analyses may introduce additional error. Our model of scientific data includes *localized error* that is estimated at every point within the domain [3,19].

We expect that the process of developing lower resolution representations will introduce additional error that increases as resolution decreases. Hence we refer to the error as cumulative error. Error information plays an important role in helping a scientist determine the appropriate level of resolution for his or her needs.

### 3.4 Data Representation

Effective exploration tools for very large data sets are best developed on a rigorous conceptual model of the data. Such a model must be accessible to both the programmer and the user and must be able to adapt to the actual data in a natural and efficient way. We now present a data model that describes scientific data that can be organized into a multiresolution hierarchy. The basic motivation for this model is to support distributed *interactive* data exploration.

A rigorous definition of a *data representation* is the formal basis of our model of the scientists data set. Although the data set represents a phenomenon defined over a continuous domain, $D$ (a *geometric* space with an infinite number of points), the data set is a finite sampling of this space. Consequently, our data representation is defined over a finite set of points $\Delta \subset D$, known as *sample points*, within the domain $D$ [8]. A sampling function $f_\Delta$ maps $\Delta$ to a subset $\Omega$ of a value space $V$ [10], denoted by

$$f_\Delta : \Delta \rightarrow \Omega \ , \tag{1}$$

with *sampling error* described by a localized error function $E_\Delta$ mapping each sample point to an error space $E$, denoted by $E_\Delta : \Delta \rightarrow E$. Formally, we define a data representation $R$ to be a quadruple:

$$R = \langle \Delta, \Omega, f_\Delta, E_\Delta \rangle \tag{2}$$

where $\Delta$ is a set of sample points in $D$ that are sampled using the sampling function with an error function $E_\Delta$, and $\Omega$ is the range of $f_\Delta$. (By convention, we use dot notation to refer to the components of a tuple. Thus, $R.\Delta$ identifies the sample points of the representation $R$.)

### 3.5 The Lattice Representation

Although the *data representation* definition is comprehensive enough to encompass most kinds of scientific data, it only represents the actual data and does not incorporate any notion of how the different data elements might be related in a grid structure. We incorporate the grid definitions into our data model by adopting and extending the *lattice* model [1]. A lattice includes a topology, $\tau$, as well as a geometry [11]. A lattice $L_k^n$ has $n$ topological dimensions that define a topological space, and $k$ attributes for a point located in that space. A 0-dimensional lattice is simply an unordered set, a 1-dimensional lattice is an ordered list, a 2-dimensional lattice lies in a plane, and so on. The lattice geometry need not have the same dimensionality as the lattice topology. For example, a 2D lattice can be mapped to a curvilinear surface that exists in three-dimensional space. Formally, a lattice $L$ consists of a data representation $R$ and a topology $\tau$; that is, $L = \langle R, \tau \rangle$. Operations on lattices include *value transformations* (e.g., normalization), *geometric domain transformations* (including affine transformations like scaling, translation, and rotation), and *topological transformations* such as mesh simplification. Operations like extension and projection can be either geometric or topological transformations (or both) depending on which components are altered.

The separation of geometry and topology in our lattice model allows us to represent both cell-based and point based datasets in a unified fashion. Either a cell-based or point-based topology can be imposed on the same dataset, allowing the user to easily switch between these two views. Similarly, data read into the system as a collection of cells can be converted into a point-based representation, and *vice versa*.

### 3.6 Simple Data Model

Our notions of *data representation* and *lattice* are sufficient to represent a gridded scientific data set, but they do not provide a representation for the phenomenon that the data set is intended to model. We now define a simple *data model* which uses the lattice to approximate the phenomenon in the domain, as well as the error. Formally, an *application data model* $M$ consists of a lattice, and functions $f_D$ and $E_C$ to approximate the data value and its associated error at every point in the domain; i.e.,

$$M = \langle L, f_D, E_C \rangle \ . \tag{3}$$

The *approximating function*, $f_D$ is normally based on the sampling function and returns a value that approximates the phenomenon in the domain; i.e., $f_D : D \rightarrow V$. *Interpolating functions* are approximating functions that satisfy the condition: $\forall d \in \Delta, f_D(d) = f_\Delta(d)$. That is, the interpolating function and the sampling function agree at each point in the sample domain $\Delta$.

A possible definition of the initial $E_C$, representing error over the entire domain, could be $f_{E_\Delta}$, which uses the approximating function to find values $E_D$ from the values of $E_\Delta$,the original sampling error function. For application data models derived from other application data models, $E_C$ is the *cumulative* error including both sampling error and the error introduced by the derivation process.

## 4 Multiresolution Data Model

Although the basic data model described above represents a very wide range of basic data sets, it is not adequate as a model for multiresolution data. A multiresolution (MR) model allows a researcher to view data using resolutions ranging from very coarse to very fine (the original data). Using a coarse resolution can vastly reduce the size of the data that needs to be stored, manipulated, and displayed. It also serves as an overview of the entire dataset, allowing the researcher to pick out regions of interest without examining the original data directly. Once an interesting region has been identified, the researcher may examine it at finer resolutions, perhaps even accessing the original data. "Drilling down" allows the researcher to examine only data of interest at fine resolution, minimizing processing and display costs.

### 4.1 Multiresolution Data Representation

The MR representation offers a tradeoff between detail and efficiency. Incorporating multiple resolution capability into the data model allows the database system to provide direct support for managing and using data at the resolution most appropriate to the immediate task.

A *reducing operator* transforms one data model into another data model, where the new representation is smaller than the old [2]. This reduction introduces additional associated localized error which must be modeled. An MR hierarchy is formed by repeated applications of reducing operations. The process is repeated a number of times until the size of the data has been reduced sufficiently or until further reductions would introduce too much error. Certain classes of wavelet functions form an ideal basis for reducing functions because of their localized error characteristics [19], but our model is also appropriate for very different kinds of data reduction techniques such as triangle mesh simplification [2].

### 4.2 Adaptive Multiresolution

An *adaptive resolution (AR) representation* allows resolution to vary within a single lattice. The resolution near a point may depend on the behavior of the sampling function, on local error, or on the nature of the domain in the neighborhood of the point. A reducing operator that behaves differently over

parts of D can define an *adaptive multiresolution representation* (AMR), which is an MR hierarchy in which each layer is an AR representation. For example, it can reduce resolution in areas with lowest error when forming the next level. It might also try to preserve resolution in areas of rapid value change and reduce resolution in less volatile areas. Because an AMR contains multiple resolutions within each level, it has the potential to achieve a representation with the same accuracy as MR using less storage. Alternatively, for a given amount of memory, it can retain increased detail and accuracy in important regions of the domain.

### 4.3 Support and Influence

Our model for MR is very general. In practice, most MR hierarchies are defined entirely by operations on the sampling set, and they often place further restrictions on a reducing function such as requiring spatial coherence. Typically, any neighboring set of sample points $S_j$ in $\lambda_i$ should map to a neighboring set of sample points $S_k$ in $\lambda_{i+1}$. $S_j$ forms the *support* for $S_k$ as shown in fig. 1. For any point $p$ there is a set of points in the next level that claim $p$ as part of their support. We call this set of points the *influence* of $p$ (see fig. 1). By building the notions of *influence* and *support* explicitly into the data model (and into the database support system), we can provide a framework for better implicit support for efficient data distribution and distributed computation.

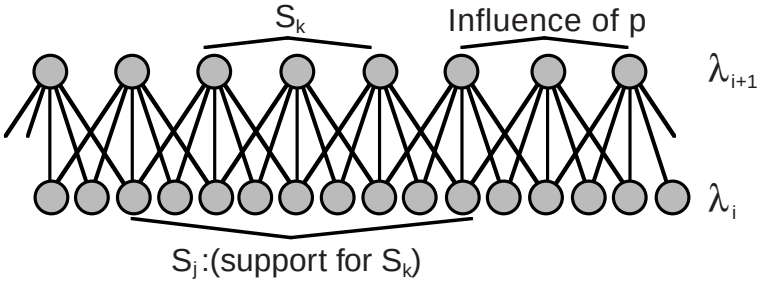


**Fig. 1.** Support and Influence

## 5 Taxonomy of Geometry and Topology

Since our representation of data must be efficient, it is worthwhile to categorize the way that data points lie in the geometry, and how they are connected in the topology. Our classification is motivated by the desire to exploit patterns within the spacing of the sample points, so the data can be represented

efficiently. The taxonomy must be able to represent both cell and point based grids and transformations between them such as Delaunay and Voronoi techniques. Our software design for the lattice representation follows from this taxonomy. We are particularly interested in how much information must be stored in order to describe the geometry and topology of the dataset.

### 5.1 Periodic Tilings and Data

The study of tilings (tessellations) has some relevance to our research since topologies often define a tiling. A review of this field can be found in [17].

If a tiling is *periodic*, then it is possible to duplicate the tiling, translate it some distance, and place it down again so that it matches exactly with the original copy. That is, the tiling consists of a number of translated repetitions of some pattern of tiles. An important and related property of periodic tilings is that there exists a subset of the space $S$ that can be repeatedly copied and translated throughout the space to complete the tiling. A minimal subset of this kind is called a *fundamental domain* or *generating region*.

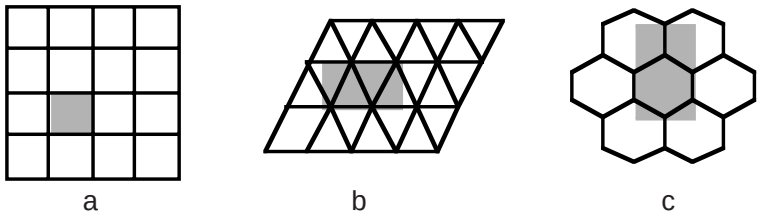


**Fig. 2.** The fundamental domains of the 2D regular tilings

A *regular tiling* is a periodic tiling made up of identical regular polygons [17]. The three tilings shown in fig. 2 are the only regular tilings for 2D space.

This approach does not explicitly distinguish between the geometry and topology components of a grid. The definition of a tiling includes aspects of topology but most concepts are geometry specific. We are adapting these ideas to our work with geometry and topology.

We use the notion of the *supercell* to represent periodic sampling topologies. As shown in fig. 3, a supercell represents a generating region for the topology, allowing the entire topology to be conceptually represented by a grid of repeated supercells while only storing a single supercell definition. If we can find where in the grid a point lies, we can very easily form a search key from the position in the grid (i.e. supercell identifier), and the position of the point within the supercell (i.e. point identifier). Such a technique promises a quick way to access a point's data given its geometric position.

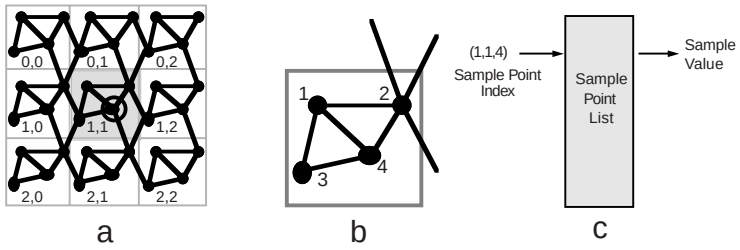


**Fig. 3.** A possible supercell implementation

### 5.2 Regular Data

Usually, for scientific data to be considered regular, it must lie within a mesh of squares or cubes, perhaps displaced by a shear operation [2]. By this definition, data points arranged in a hexagonal fashion, as in case *c* of fig. 2, would not be considered regular, though the first two would. However, this may only be the case because of the ubiquity of array storage. Researchers tend to think of regular data as any data that can be stored very easily in an array.

It should be possible to develop a rigorous definition of regular data. Besides being regular in the mathematical sense, the patterns in fig. 2 have an interesting property: if we store the vertices (sample points) in an array, it is possible to map a point's array indices to its locations in both the geometry and topology without using any other information. Since this property is of immediate interest to designers of scientific databases, it might serve well as a definition of regular scientific data.

### 5.3 Classifying Irregular Data

With irregular data, it is not possible to map array indices to a location in geometric space without using extra information, if at all. Of course, arrays may still be used to merely store the data points. Figure 4 shows examples of irregular data. In fig. 4.a, there is no way to map indices to a geometric location without referring to the spacing between the rows and columns, which varies for each row and column. Therefore, the mapping between indices and geometry must take this spacing as another parameter, i.e. as "extra information". The situation in fig. 4.b is even worse. Here, there is no pattern whatsoever to the position of the sample points, so a simple mapping from indices to geometry is out of the question.

We further classify irregular data according to how much information is required to represent the pattern of sample points within the domain. In fig. 4.a, points are lined up in rows and columns, so we only need to store the

spacings for each row and column. The space required to store this information is proportional to the number of rows plus the number of columns. In fig. 4.b, there is no pattern whatsoever to the sample points, so we must store coordinates for each individual point. Here, the space required is proportional to the number of points. Of course, it is possible to have datasets that combine these attributes, behaving in different ways along different dimensions.

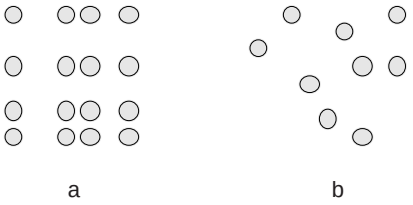


**Fig. 4.** Irregular data

## 6 Datasources

Lattices provide the scientist with a conceptual view of his or her data that should be consistent with the operations that need to be applied to the data. In principle, this conceptual view will be reflected in the organization of the physical data. In practice, however, this is often not feasible. The scientist may need different views of the same data and the data may be too large to replicate and reorganize to match each desired view. In general, multisource data and distributed computing require sophisticated ways of dividing large files into smaller pieces while maintaining a simple view of the distributed data.

### 6.1 Mapping Lattices to Data

A lattice is able to map locations in the geometry to locations in the topology. It remains to map topological locations to offsets in file or network streams. A *datasource* provides the lattice with a single, unified view of multisource data. This simplifies the mapping from topological locations to file and network stream offsets.

Some datasources are directly associated with a local file or remote source, and are known as *physical* datasources. Other datasources are *composite*, meaning they are made up of more than one component datasource. For example, a datasource that performs an attribute join would be composite. It is possible to perform very complex operations by combining several datasources together in a tree structure, with the *root datasource* at the top of the tree providing the lattice with a simplified view of the data.

### 6.2 Datasource Model

A datasource can be modeled as an n-dimensional array containing $\delta$, a subset of the set of lattice sample points $\Delta$. We think of arrays as an *index space* $d$ paired with a collection of associated data values. An index space can be expressed as the cross product of several indices, each defined as a finite subset of the integers:

$$I_1 \times I_2 \times \cdots I_n \tag{4}$$

where each $I_k$ is an integer in the range $[a_k \ldots b_k]$. The dimensionality of a datasource's index space may or may not match the dimensionality of the lattice domain $D$. If these dimensionalities do match, then the neighborhood relationships present in the lattice may be reflected in the adjacencies present in the underlying storage. In other cases, there is no simple pattern in the distribution of $\Delta$ in $D$, so the lattice topology must map points from $D$ into the underlying index space.

### 6.3 Physical Datasource

A *physical datasource* is a simple datasource that is directly connected to a file or network stream that contains sample points. While the stream is in reality a one dimensional entity, a physical datasource may have an index space that is n-dimensional. In this case, the physical datasource is responsible for mapping the n-dimensional view to the underlying data. This mapping can be expressed as a function $m$ that maps an index space to a single index $I_f$ used to access the actual data block.

$$m : I_1 \times I_2 \times \cdots I_n \rightarrow I_f \tag{5}$$

For example, consider a file of three dimensional array data. Perhaps the layers are stored in order of increasing z value, and each layer is stored in row major order. The physical datasource is responsible for mapping the x, y, and z values of its data space to a file index, which is really an offset from the beginning of the file. Of course, a single file may contain many attributes, so a physical datasource should be capable of returning all values corresponding to a location in the index space.

### 6.4 Blocked Datasource

A *blocked datasource* is a composite datasource in which the index spaces of the component datasources are joined together in contiguous, non-overlapping fashion to form a single index space[1].

$$d = \bigcup_{i=1 \ldots k} d_i \text{ where } \forall i, j : i \neq j \rightarrow (d_i \cap d_j = \oslash) \tag{6}$$

[1] As defined here, this is an outer natural join. It is possible to relax the contiguity and non-overlap constraints, but this is beyond the scope of this paper's goals.

For example, consider two datasources *ds1* and *ds2* that might represent two contiguous satellite image files, as shown in fig. 5. Their index spaces can be joined together in the fashion shown by *ds3*, a blocked datasource, producing a single index space that can be manipulated as a single entity. Of course, a blocked datasource can have an arbitrary number of component datasources, allowing large amounts of data to be viewed as a single entity, but stored and accessed in a distributed fashion.

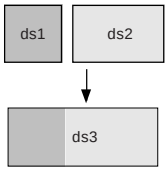


**Fig. 5.** Two datasources joined by a blocked datasource

### 6.5 Attribute Join Datasource

An *attribute join datasource* is a composite datasource for which each point in $\delta$ is composed of attributes taken from two or more component datasources. If $A$ is the attribute set of an attribute join datasource, then we say:

$$A = \bigcup_{i=1...n} A_i \tag{7}$$

where $A_i$ are the attribute sets of the component datasources. For example, suppose *ds1* is a datasource with attributes {*salinity, pH, oxygen*} and *ds2* is a datasource with attributes {*temperature, depth*}. If these two datasources are combined by an attribute join datasource *ds3*, then each point in the index space of *ds3* has attributes {*salinity, pH, oxygen, temperature, depth*}. Such an operation is particularly useful when data has been organized into separate files, perhaps because it was gathered by different instruments.

## 7 Datasource Implementation

In this section we describe the primary elements of our prototype implementation and give a simple example of how the system processes multisource data for both rectilinear and unstructured lattices.

### 7.1 Implementation Components

Figure 6 depicts the key components of the implementation of our model. Lattice methods provide the functionality by which scientific applications interact with lattice data. Each lattice has associated Geometry and Topology

objects that map sample points of the lattice to a computational space in the form of an N-dimensional array implemented as a DataSource object. We use an array because physical storage is easily conceptualized as an array. The mapping may be simple as when the sample grid forms a regular pattern over the domain so that each sample point can be naturally associated with a unique array element. The mapping is more complex for unstructured sampling patterns.

The DataSource transforms computational space to low level file or network URL requests. When all data appear in one file or one internet data server, the DataSource may be implemented directly as a PhysicalDataSource which maps an N dimensional data identity (index position) onto a 1 dimensional storage device. Our DataSource is designed to be flexible enough to integrate data from multiple physical data sources. We allow both spatial and attribute joins. The BlockedDataSource is composed of multiple sources for separate regions of the index space. The AttributeJoinDataSource combines attributes from multiple datasources. Blocked and attribute join data sources can be combined and nested to reflect multiple files and/or network URLs.

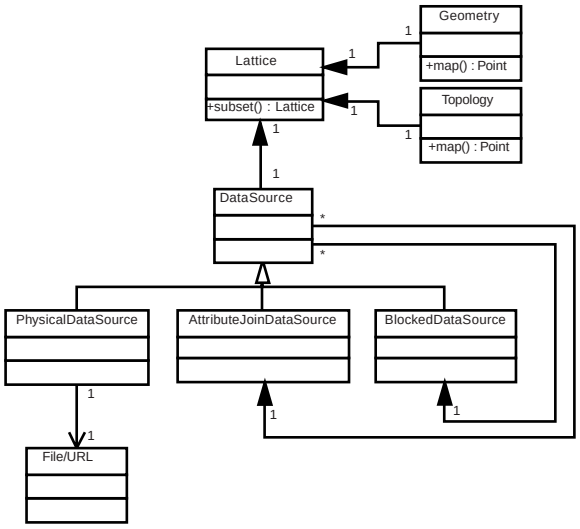


**Fig. 6.** A UML representation of key components of our system

### 7.2 An Example Scenario

In this section we discuss two running examples to illustrate the concepts of the lattice, datasource, and the relationship between them. Our first example involves rectilinear data in which there is a simple mapping between the

location of a point in the lattice geometry and its location in the datasource index space. The second example involves unstructured data for which there is no such simple mapping, forcing the lattice topology to play a greater role.

**A Rectilinear Lattice** Consider the two dimensional lattice in the diagram below. The sample points are obviously placed in a regular fashion within the geometry. It is most likely that the experimenter will choose to use a rectilinear topology for such data as shown in fig. 7.b, although it should be understood that a different choice could be made. Lets suppose that each point has attributes {*time, carbon, nitrogen, oxygen*}.

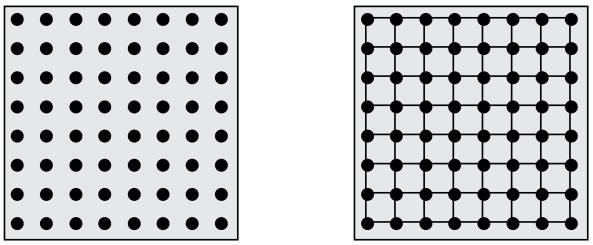


**Fig. 7.** A 2D Rectilinear Lattice: **(a)** geometry only, and **(b)** with topology imposed

The query `lattice.datum(p)` asks for a value corresponding to a point $p$ in the domain, which is perhaps the simplest kind of lattice query. The lattice topology is responsible for associating a point in the geometry with a location in a datasource index space. Since we know there is a simple mapping from geometric location to index space, it is easy to check whether $p$ is a sample point, or sufficiently close to one that it can be treated as such. In this case, the Topology computes the datasource index for the point and then issues the query `ds.datum(new TwoDIndex(i,j))` to the root DataSource, which then returns the correct value for the sample point.

If the query point does not correspond to a sample point, the lattice approximating function must be used to generate an approximate value. To do so it must be given the values of nearby sample points. Because the topology closely reflects the Geometry, it is quite efficient to use the lattice topology to select the four sample points that surround $p$. Their indices are then placed into a list, after which the query `ds.datumList(indexList)` is issued to the root DataSource. The values returned can then be given to the approximating function to return a computed value.

**An Unstructured Lattice** The process for an unstructured lattice is similar overall, except that there is no simple mapping between the location of

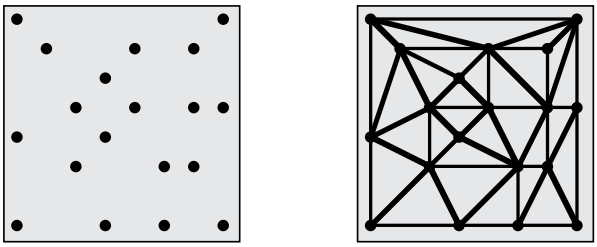


**Fig. 8.** A 2D Unstructured Lattice: **(a)** geometry only, and **(b)** with topology imposed

sample points in the geometric space and their location in the datasource index space. A 1D datasource is often appropriate for unstructured data, effectively treating the data as a list of sample points. The lack of a simple mapping forces the topology to do more work when computing the datasource indices for sample points near the query point in the geometric space. In fact, a multidimensional access method [5] such as a quadtree could be used to allow efficient search through the geometric space for nodes in the topology graph that correspond to nearby sample points. These nodes will contain the datasource index of the sample point they represent. Once nearby sample points have been found, it is relatively easy to navigate the Topology graph to find the set of points nearest the query point, and then send an array of indices to the root datasource to have the corresponding values retrieved. As with the rectilinear example, the query `ds.datumList(indexList)` will be sent to the root datasource after the indexes of neighbor points have been put into the indexList.

**An Example Datasource with Rectilinear Data** Now that we have seen how a lattice translates geometric queries into datasource queries, we examine how datasource queries are satisfied. Suppose the data consists of one attribute that is stored in a single file organized as a 2D array and the other three attributes are stored in four files that represent the four quadrants of the 2D array. This structure is shown by the DataSource tree in fig. 9. The index spaces of the DataSources in this example are always two dimensional. The four physical DataSources in the lowest level of this tree are associated with network streams communicating with remote machines. Lets assume that their attribute sets are all {*carbon, nitrogen, oxygen*}. The BlockedDataSource in the middle level joins the index spaces of its components together, but shares their attribute set. The PhysicalDataSource in the second level is associated with a local file containing the single attribute {*time*}. The two DataSources on the second level have identical index spaces

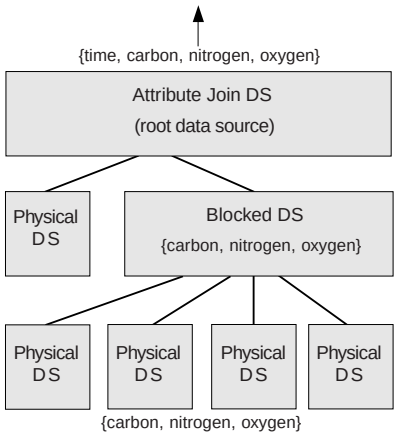


**Fig. 9.** An example DataSource tree

and are combined using an AttributeJoinDataSource. This topmost DataSource is the root DataSource that communicates with the Lattice itself.

Let's consider the query `ds.datum(new TwoDIndex(i,j))` given to the root DataSource by the Lattice. The parameter is an index for the root DataSource. Since this DataSource is an attribute join, it sends the same query to both of its component DataSources. No transformation is necessary because the component DataSources have the same index space as the root DataSource. Of course, the PhysicalDataSource in the second level maps the two dimensional index to a one dimensional file offset and then return values for its attribute. The BlockedDataSource has more work to do. It must decide which of its component DataSources contains the index and then transform it into the index space for that DataSource. After retrieving the result from the chosen physical DataSource, the BlockedDataSource sends the three attributes up to the root DataSource, which joins them with the one already retrieved from the physical DataSource. Finally, the root DataSource returns the four attributes to the Lattice.

**An Example DataSource with Unstructured Data** For unstructured data, the principal difference at the datasource level is that the index spaces of the various DataSources do not match the dimensionality of the Lattice. Since unstructured data is often stored as a list of points, the datasource index space may well be simply one dimensional.

To continue our unstructured data example, lets assume that the DataSource tree is the same as the one depicted in fig. 9. The root DataSource still receives the query `ds.datum(index)` from the Lattice, but now the index is one dimensional. The BlockedDataSource picks which PhysicalDataSource in the bottom level to query based on the index value, retrieving {*carbon,*

*nitrogen, oxygen*} from the appropriate component. The PhysicalDataSource in the second level retrieves {*time*}, and finally the root DataSource joins the four attributes together. Notice that except for the dimensionality of the index space, the tree works in a fashion which is identical to the rectilinear case.

## 8 Performance Issues

Systems that handle very large datasets should be designed to minimize redundant data access and to take advantage of any structure in patterns of access. It is also important to avoid unnecessary duplication of data.

Our prototype system allows us to experiment with and evaluate techniques for minimizing access costs. We are particularly interested in finding ways to reduce overhead associated with distributed multisource data.

### 8.1 Lazy Evaluation

The concept of a datasource fits naturally with lazy evaluation. This is especially apparent if we think of a tree of datasources as a description of a data file that does not physically exist. When the tree is built, no data is actually processed until the Lattice begins asking for sample points. Even at this point, the file that the Lattice sees is only conceptual. Sample points are assembled from component datasources but the entire file is never materialized.

We have already added a small enhancement to our prototype system by allowing datasource `datum()` queries to take an argument that specifies which attributes should be retrieved. In practice, this information is sent down to the physical datasources so that only desired attributes are read and processed by the rest of the datasource tree. Though conceptually simple, this enhancement could greatly reduce the volume of data that is processed in situations where each sample point has a large number of attributes.

We also have defined a subset operation to both the lattice and datasource levels. At the lattice level it is useful to specify a new lattice as a subset of an existing one. However, it is not practical to duplicate the lattice sample points; it is much more efficient to implement a new *subset datasource* that presents the new lattice with a subset of the original data without ever materializing an actual subset of the original. Of course, this scheme relies on the fact that much scientific data is read-only. More complex mechanisms are needed to support multiple lattices writing to the same data.

### 8.2 Caching and Prefetching

Future versions of our prototype system will incorporate caches as a primary tool for avoiding redundant access to data.

When data is accessed in a regular pattern, it is possible to predict which elements will be required in the near future. Processes that iterate over a lattice in a predictable fashion can be greatly accelerated by prefetching data that is likely to be needed. Prediction may be straightforward with rectilinear data since iteration will often proceed through rows and columns in a regular fashion [13]. Effective prefetching of unstructured data is harder but can be done using a topological or geometric *neighborhood*.

In fact, a *flexible neighborhood query* can serve as the basis for prefetching for both structured and unstructured data. With this kind of query, a lattice would ask its root DataSource to return two sets of points, a *primary* set and a *secondary* set (such as a neighborhood). The DataSource must return the primary set and may return some or all of the secondary set if it is convenient. For example, a DataSource might return a partial result consisting of the primaries and readily available secondaries. Other secondaries may later arrive and can be cached for future queries.

The Topology and Geometry provide valuable information for identifying appropriate secondary data points based on the nature of the processing being done.

## 9 Conclusion

We have developed a formal data model for describing distributed multiresolution multisource scientific data sets. The lattice provides the logical view of data as it appears to the scientist. The DataSource is the principal mechanism for an efficient implementation of multisource datasets. Lattices also include an explicit definition of topology which allows us to represent a wide variety of grid structures. The topology uses the datasource representation to map data points to physical locations. The entire model serves as the basis for a scientific data management support environment that can provide nearly transparent access to distributed multisource data.

We have an initial implementation of a prototype system to support the principal features of our data model. This implementation allows a scientist to describe a single dataset that represents a unified view of data that is physically stored in multiple datasets.

Although our current prototype is designed to include caching and prefetching, these features have not yet been implemented. Similarly, our current support for unstructured data is minimal and we need to integrate multiresolution and adaptive resolution functionality into the prototype.

## References

1. R. Daniel Bergeron, Georges G. Grinstein, "A Reference Model for the Visualization of Multi-Dimensional Data", Eurographics '89, Elsevier Science Publishers, North Holland, 1989

2. Paolo Cignoni, Claudio Montani, Enrico Puppo, Roberto Scopigno, "Multiresolution Representation and Visualization of Volume Data", IEEE Trans. on Visualization and Computer Graphics, Volume 3, No. 4, IEEE, Los Alamitos, CA, 1997
3. P. Cignoni, C. Rocchini and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces", Computer Graphics Forum, Vol. 17, No. 2, Blackwell Publishers, Oxford, UK, 1998
4. Mark de Berg, Katrin T.G. Dobrindt, "On Levels of Detail in Terrains", Graphical Models and Image Processing 60:1–12, Academic Press, 1998
5. Volker Gaede, Oliver Günther, "Multidimensional Access Methods", ACM Computing Surveys, Vol. 30, No. 2, ACM, New York, 1998
6. R.B. Haber, B. Lucas, N. Collins, "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids", Proceedings of IEEE Visualization 91, San Diego, CA, 1991
7. W.L. Hibbard, C.R. Dyer, and B.E. Paul, "A Lattice Model for Data Display", Proceedings of IEEE Visualization '94, IEEE, Washington, DC, 1994
8. W.L. Hibbard, D.T. Kao, and Andreas Wierse, "Database Issues for Data Visualization: Scientific Data Modeling", Database Issues for Data Visualization, Proc. IEEE Visualization '95 Workshop, LNCS 1183, Springer, 1995
9. D.T. Kao, R. Daniel Bergeron, Ted M. Sparr, "An Extended Schema Model for Scientific Data", Database Issues for Data Visualization, Proceedings of the IEEE Visualization '93 Workshop (LNCS 871), Springer, Berlin, 1993
10. D.T. Kao, M.J. Cullinane, R.D. Bergeron, T.M. Sparr, "Semantics and Mathematics of Scientific Data Sampling",in Wierse, Grinstein and Lang (Eds.), Database Issues for Data Visualization, LNCS 1183, Springer-Verlag, Berlin, 1996
11. D.T. Kao, "A Metric-Based Scientific Data Model for Knowledge Discovery", Ph.D. Thesis, University of New Hampshire, Durham, 1997
12. D.T. Kao, R.D. Bergeron, T.M. Sparr, "Efficient Proximity Search in Multivariate Data", 10th International Conference on Scientific and Statistical Database Management, Capri, Italy, 1999
13. Heng Ma, "Remote Transformation and Lattice Manipulation", Masters Thesis, University of New Hampshire, Durham, NH, 1992
14. John L. Pfaltz, Russell F. Haddleton, James C. French, "Scalable, Parallel, Scientific Databases", Proceedings 10th International Conference on Scientific and Statistical Database Management, IEEE, Los Alamitos, CA, 1998
15. Richard J. Resnick, Matthew O. Ward, and Elke A. Rundensteiner, "FED – A Framework for Iterative Data Selection in Exploratory Visualization", Proceedings of Tenth International Conference on Scientific and Statistical Databases, IEEE Computer Society Press, Los Alamitos, CA, 1998
16. Philip J. Rhodes, R. Daniel Bergeron, Ted M. Sparr, "A Data Model For Distributed Scientific Visualizations", Proceedings of the Scientific Visualization Conference:DAGSTUHL 2000 (To be published)
17. Doris Schattschneider, Marjorie Senechal, "Tilings", Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, 1997
18. D. Speray, S. Kennon, "Volume Probes: Interactive Data Exploration on Arbitrary Grids", Computer Graphics, Vol. 24, No. 5, ACM, 1990
19. Pak Chung Wong, R. Daniel Bergeron, "Authenticity Analysis of Wavelet Approximations in Visualization", Proceedings of IEEE Visualization '95, IEEE Computer Society Press, Los Alamitos, CA, 1995

# Index