The ATree: A Data Structure to Support Very Large Scientific Databases*

Pedja Bogdanovich and Hanan Samet

Computer Science Department and Center for Automation Research and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA

Abstract. The datasets generated by satellite observations and supercomputer simulations are overwhelming conventional methods of storage and access, leading to unreasonably long delays in data analysis. The major problem that we address is the slow access, from large datasets in archival storage, to small subsets needed for scientific visualization and analysis. The goal is to minimize the amount of storage that has to be read when a subset of the data is needed. A second goal is to enhance the accessibility of data subsets by applying data reduction and indexing methods to the subsets. The reduced format allows larger datasets to be stored on local disk for analysis. Data indexing permits efficient manipulation of the data, and thus improves the productivity of the researcher. A data structure called the ATree is described that meets the demands of interactive scientific applications. The ATree data structure is suitable for storing data abstracts as well as original data. It allows quick access to a subset of interest and is suitable for feature-based queries. It intrinsically partitions the data and organizes the chunks in a linear sequence on secondary/tertiary storage. It can store data at various resolutions and incorporates hierarchical compression methods.

1 Introduction

Present day and future supercomputer simulations and observations by satellites and other monitoring devices produce very large data sets. These datasets are overwhelming conventional methods of storage and access, leading to unreasonably long delays in data analysis. Because of this, the speed of processors is no longer an issue in scientific applications—the management of terabytes of data is the major bottleneck.

Future hardware technology developments will certainly help the situation. Data transfer rates are likely to increase by as much as an order of magnitude, as will tape and cartridge capacities. However, new supercomputers and massively-parallel processor technologies will allow scientists to calculate at finer spatial

* This work was supported in part by the Department of Energy under Contract DEFG0592ER25141, the National Aeronautics and Space Administration under Fellowship Grant NGT-30130, and the National Science Foundation under Grant IRI-9712715.

P. Agouris and A. Stefanidis (Eds.): ISD'99, LNCS 1737, pp. 235–248, 1999.

and temporal resolutions, and thus generate more data. Most of the data generated by models and experiments must reside on tertiary devices, and only a subset of that data will usually be of immediate interest. The effective management of large amounts of scientific data will therefore be an ongoing concern.

Interactive analysis and visualization applications frequently require rapid access to relatively small subsets of this data. Current commercial database management systems (DBMSs) provide inadequate support [15]. The primary reasons are: Conventional DBMSs do not adequately support data abstractions such as spatio-temporal data arrays. The indexing methods they provide are inappropriate for this type of data, making proximity queries, and operations based on spatial and temporal ranges, very expensive. Current DBMSs do not support tertiary storage. They only support secondary storage (i.e., disks), which is not sufficient for the massive data generated and analyzed in scientific applications.

The initial thrust of this research is to solve problems such as those faced by climate model researchers being unable to easily access previously stored data. The methods described here may benefit various scientific and engineering applications in areas identified as Grand Challenges. Examples of areas that use large-scale multidimensional simulation programs and analysis methods are seismic studies, oil reservoir modeling, climate modeling, fluid turbulence, vehicle dynamics, and hazardous waste management. This research uses global climate modeling as a concrete example of an application area that can take immediate advantage of our solutions and determine their effectiveness. We fully expect to be able to generalize our work to other Grand Challenge areas, or to any application or experimental area requiring fast access to small subsets within large collections of scientific data.

The major problem we wish to address is the slow access to small subsets of large datasets in archival storage needed for visualization and analysis. The goal is to minimize the amount of storage that has to be read when a subset of the data is needed. A second goal is to enhance the accessibility of data subsets by applying data reduction and indexing methods to the subsets. The reduced format will allow larger datasets to be stored on local disk for analysis. Data indexing will permit efficient manipulation of the data, and thus improve the productivity of the analyst.

In this paper, we introduce a data structure called the ATree which achieves the goals listed above. Section 2 describes typical raster data, as well as the concept of abstracts. Section 3 describes the ATree data structure, Section 4 describes our algorithms, and Section 5 describes the user interface.

2 Raster Data

Our research focuses on raster data. Raster data is used to represent multidimensional vector field data, sampled at discrete points. Such data is generated by satellite observations and supercomputer simulations.

Raster data is the most voluminous data type encountered in Global Change Research. For example, a single Landsat Thematic Mapper image is 300 megabytes. Other satellites beam down on the order of terabytes of data per day [5].

Raster data is also generated by a climate modeling simulations. A climate simulation model is run for a simulation period of 30 years. At every 3–6 hours of simulation time the state of the model is dumped out. The resulting dataset consists of a number of *variables*. A variable is a multidimensional array of values of the same type. More precisely, a variable is a multidimensional vector field, most often a scalar field. That is, each variable is a multidimensional array corresponding to a certain spatio-temporal domain. The first characteristic of the data is that the number of dimensions is quite small, usually at most four. Typical dimensions are $longitude \times latitude \times height \times time$. Typical variables are air temperature and wind velocity defined over these dimensions. The number of variables in a dataset is also relatively small, up to several hundred. However, the volume of data for each variable is large because the spatio-temporal extent of the data is large. In the case of climate modeling data, this results from a large number of samples in the temporal coordinate.

A typical dataset contains from several gigabytes to several terabytes of data. Hence, this data is typically kept on tertiary devices (Exabyte tapes, OD's, etc.). In order to analyze the data, it is downloaded to a local disk.

Operations on raster data fall into one of two categories:

Spatial extraction: extracting a subset of values corresponding to a hyper-rectangular region of multidimensional space (i.e., subsetting and slicing). Examples of higher-level operations that rely on spatial extraction are browsing, data visualization, and statistical analysis of data. Such operations are also known as window queries (e.g., [2]).

Feature queries: i.e., value searching. Example: "find all regions which contain a hurricane". Typically this type of query accesses large portions of the data sets, but the answers are small.

Figure 1 shows a three-dimensional set of raster data.

Abstracts

In order to deal with terabyte data sets, we introduce a storage management technique called *abstracts*. Other Grand Challenge projects such as Sequoia 2000 [16] have also proposed the use of abstracts [7]. Abstracts involve extraction of certain "essential" parts of the original data. Abstracts can be seen as precomputed answers to anticipated queries, or precomputed data that helps answer a class of queries. Abstracts are several orders of magnitude smaller than the original data sets.

A data set can have multiple abstracts associated with it, each corresponding to a different "essential" part of the data set. Thus, each abstract is a different view of the data. What the abstracts are really depends on the data and on the *context* in which the data is used. Note that an abstract need not produce the same kind of information as does the original data. Another way to view an abstract is as a secondary index into the original data since abstracts let us quickly position into the original data.

Abstracts have many benefits. If a query can be answered from an abstract, then the bandwidth increases and the response time decreases. Since abstracts are several orders of magnitude smaller than the original data set, abstracts can be kept in secondary storage instead of in tertiary storage. Hence, the speedup is cumulative (smaller size + faster storage device), and may even be greater since caching can now be effectively employed. Compressing datasets using lossy compression methods, such as hierarchical compression [4], is one example of data abstraction. Another example is the use of multiresolution data structures [Salem92]. In this case, a collection of abstracts is represented at different levels of detail.

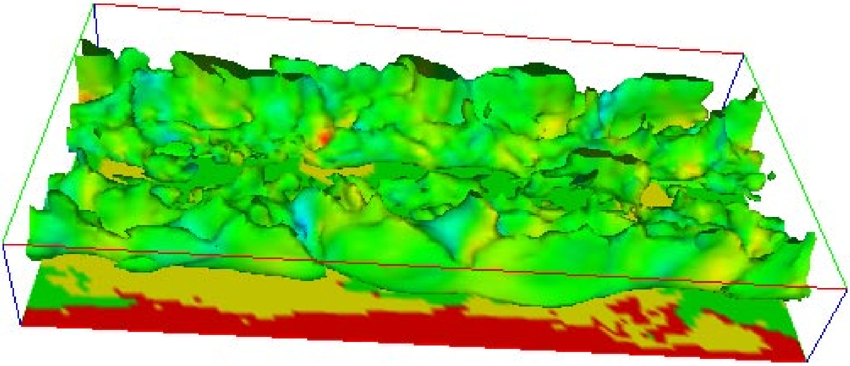

Fig. 1. Iso-surface of one slice in the temporal dimension of a wind-velocity vector field (raster data) generated by a climate modeling simulation.

3 The ATree Data Structure

The ATree data structure is an extensible data structure suitable for the storage of original data sets, as well as abstracted data sets. The main features of the ATree are:

- The ATree data structure is a multidimensional indexing data structure. Spatial indexing allows quick access to data based on its spatial location.
- Data is partitioned into chunks according to the principle of spatial locality. The size and shape of the chunks is chosen according to the expected usage of the data.
- Data chunks are organized (laid out) on archival storage according to the expected usage of the data. For example, if Morton codes [8, 12] are used, then data chunks are stored on secondary/tertiary storage devices according to the principle of spatial locality.
- The data structure allows for hierarchical compression techniques to be applied (both lossy and non-lossy). Note that unlike traditional compression techniques, this data structure allows for direct indexing into compressed data. This means that only chunks of data that are accessed by the user are decompressed.
- The ATree data structure can store multiresolution data.
- The data structure allows for non-spatial information to be included, which facilitates quick retrieval of data based on non-spatial attributes (e.g., crop types).

The ATree data structure is an extension of the region quadtree data structure [13, 12]. We describe the ATree data structure below.

3.1 Subdivision Scheme

Unlike the region quadtree which always subdivides quadrants into exactly four disjoint sub-quadrants, the blocks corresponding to the region nodes of the ATree, called *orthants*, are subdivided along pre-specified coordinates at each level of the tree. (The same coordinates are subdivided for all the orthants at the same level.) At least one coordinate has to be subdivided at each level of decomposition. When an orthant is subdivided along a coordinate, it is subdivided into two halves. Figure 2 represents a four-level deep ATree.

For example, we can implement KD-trees [3] by defining subdivision of orthants to be performed on different coordinates (cyclically) at each level. Region quadtree-like decomposition can be achieved by an ATree when the orthants are subdivided along all coordinates at each level.

The benefits of this approach in comparison with the region quadtree approach are as follows:

- Region quadtrees are not suitable for storing data with non-square extents since the leaf nodes also have non-square extents. This is desirable in cases where one coordinate has many more samples than the others. The ATree subdivision scheme allows data with non-square extents to be stored.

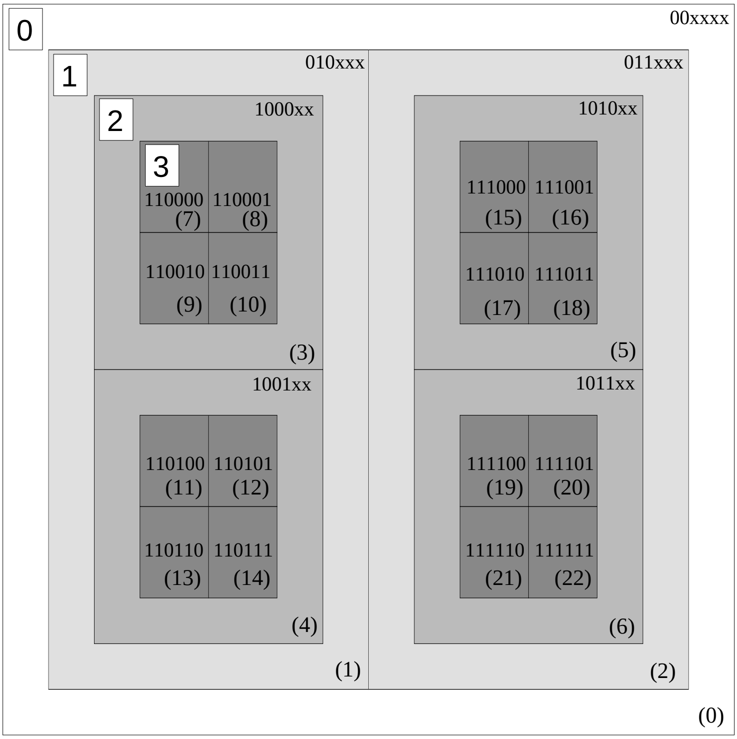

Fig. 2. Shape of a four level ATree: The root orthant is subdivided along the x-coordinate, the first-level orthants are subdivided along the y-coordinate, and the second-level orthants are subdivided along both the x and y coordinates. Binary numbers represent a location code encoding of the orthant, and the numbers in parentheses represent orthant ordinal numbers for the encoding used.

- Multidimensional generalizations of the region quadtree are impractical even for moderately high dimensions due to the high fan-out factor (2^d). The subdivision criterion used by the ATree provides fine control over the fan-out factor at each level of the structure.
- Orthants specify how data is chunked. By controlling the subdivision scheme, we can control the shape of the data chunks.

3.2 Location Codes

Since the data is to be stored in memory, it is often necessary to linearize it so that is is ordered and can be accessed. In order to linearize multidimensional data, a location code scheme is used. The linearization process maps hyperrectangles to integers. We use the function lc. In particular, $k = \mathrm{lc}(r)$ where r is a hyperrectangle corresponding to some orthant in the subdivision, and k is the orthant's number. The function lc has to be 1-1 and onto since lc^{-1} must be defined.

Location codes based on Morton order (also known as Z-order [9]) linearize multidimensional space so that the orthants which are spatially close tend to have their location code numbers close. However, the ATree data structure allows for any consistent location code scheme to be used. By using different location code schemes, we can specify how orthants (data chunks) are laid out on archival storage.

For example, consider a multiresolution representation of data in which we store both leaf and non-leaf nodes. Nodes deeper in the tree refine the data from higher levels. In this case we might want to have a breadth-first arrangement of nodes in order to have data clustered on a per-level basis. This is easily achieved by using a location code scheme which first encodes the depth of the node and then its bit-interleaved top-left corner. Figure 2 illustrates one such encoding. The encoded values of orthant nodes are given as binary numbers. Symbol 'x' represents a "don't care" value. The depth is encoded in the first two bits and the remaining bits form the Morton encoding for the bottom-left orthant corner. Note that orthants which lie on the same level have ordinal numbers which are close.

3.3 Transformation Function

A transformation function specifies what is actually stored with each ATree orthant node. It is applied to both internal and leaf nodes of the tree. By storing data in non-leaf orthant nodes we get a pyramid [17], termed the sc APyramid, variant of the ATree data structure. This allows data to be stored at different resolutions—orthants near the root of the tree contain low resolution data, and as we descend the tree, the data is refined further.

Note that in order to facilitate data retrieval, the inverse of the transformation must exist. Let $d = t(o)$, where o is a subset of the original data corresponding to some orthant, t is a transformation function, and d is the data returned by the transformation function, i.e., the data that is actually stored on the disk. Now, let t^{-1} be the inverse of the transformation function, that is, the function that takes raw data retrieved from the disk and returns data *corresponding* to the original untransformed data. Ideally, $o = t^{-1}(t(o))$ but this need not to be the case, such as, for example, in the case of lossy compression.

The transformation function can also specify non-spatial data that is to be stored with each orthant. This information can be used to aid in answering non-spatial queries.

3.4 Subdivision Criterion

The subdivision criterion specifies how deeply we go in subdividing orthants. If the ATree does not store internal nodes and if the transformation function is the identity function, then the subdivision criterion effectively specifies the size of data chunks stored on the disk. In this case, one of the approaches that we employ is to stop subdividing space as soon as the data associated with the orthant is of a certain size (leaf-orthant-data-size). In order to maximize performance when data is stored on secondary storage devices, we will use the size of a virtual memory page size as the size of the leaf orthant data. For tertiary devices, bigger chunks are appropriate, e.g., we may use the size of the platter.

As another example, consider a transformation function which is a lossy transformation. Assuming that transforming data corresponding to smaller spatial extents leads to more accurate transformed data, a subdivision criterion might inverse-transform the data returned by the transformation function and compare it with the original data. If the accuracy is greater than a certain prescribed accuracy, the criterion is not satisfied. In this way we can directly control the level of accuracy achieved.

Figure 3 shows a two-dimensional ATree decomposition of map data. We used the following subdivision criterion: we stop subdviding orthant nodes as soon as all the values within a node are equal, all the way to the pixel size, if necessary (as in region quadtrees). Typically we do not subdivide data that to such a fine level of detail. In practice, we use some of the subdivision criteria described above.

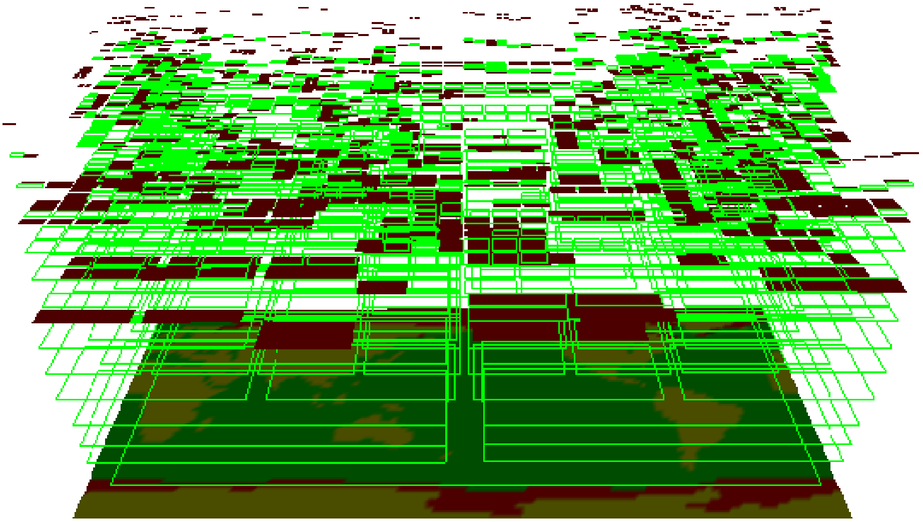

Fig. 3. Region quadtree-like subdivision criterion: Stop subdividing orthant nodes as soon as all the values within a orthant are equal.

4 Algorithms

In this section we describe some ATree algorithms.

4.1 Build

The build operation takes the raw data in some standard format (DRS, NetCDF, HDF) and builds an ATree. The input parameters for the build operation are also the parameters that define the ATree:

1. Shape: defines the subdivision scheme to be used. Recall that the subdivision scheme determines the spatio-temporal regions to which the subdivision criterion and the transformation function will be applied.
2. Location code scheme: defines the layout of the data on archival storage. Recall that the location code scheme should be chosen according to the expected access pattern of the dataset. For example, for multiresolution data structures we want some breadth-first location codes, since we want to cluster data on a per-level basis.
3. Transformation function: defines what is actually stored with each node, e.g., the data reduction technique that is being used. This is determined by the actual data and its intended scientific use.
4. Subdivision criterion: defines the level of detail stored in the data. This is determined by the required level of detail or accuracy.
5. Optional metadata function: other information that we may wish to associate with a node.

Below, we describe the top-down and bottom-up build algorithms.

Top-Down Build Algorithm. The idea behind the top-down build algorithm is to start from the root orthant node and then recursively subdivide as needed. The top-down build algorithm is given in Figure 4.

Starting from the root orthant node,

1. Apply the subdivision criterion to the orthant node.
2. If the criterion is not satisfied, then the orthant node is a leaf node. In this case, apply the transformation function to the subset of data corresponding to the orthant node, and store the data returned.
3. If the criterion is satisfied, then the orthant node is an internal node. Subdivide the node and recursively apply the procedure to the descendant nodes. In the case of an APyramid, data is stored with internal nodes as well.

Fig. 4. Top-down build algorithm.

The top-down approach may be impractical since it may be hard to apply a transformation function to orthant nodes with large spatio-temporal extents (that is, nodes near the top of the tree) since the amount of the data associated with these nodes is large. However, the top-down build algorithm may be used for the APyramid, since the transformation function has to be applied to all the orthant nodes anyway.

Bottom-Up Build Algorithm. The idea behind the bottom-up build algorithm is to start from the deepest possible level of the tree and merge. The bottom-up build algorithm is given in Figure 5.

Start from the deepest possible level of the tree and generate all the leaf orthant nodes. The process proceeds by repeating the following procedure for each level of the tree:

1. For every mergeable group of sibling nodes, generate a parent node.
2. Apply the subdivision criterion to the parent node. If the criterion is not satisfied, then remove all the descendant nodes.

(The algorithm is implemented as a post-order tree traversal.)

Fig. 5. Bottom-up build algorithm.

4.2 Subset Algorithm

The subset operation extracts data from an ATree that corresponds to a given extent. In addition to the subset extent, the subset operation also requires an *accuracy test function* to be specified. The accuracy test function returns **TRUE** if the data stored with a current orthant is accurate enough; otherwise, it returns **FALSE**, meaning that we need to go deeper in the tree. The subset algorithm is given in Figure 6.

Starting from the root orthant node,

1. If the orthant's extent does not intersect the subset's extent, then return.
2. If there is no data stored with the orthant node, then skip to step 4.
3. Retrieve the data stored with the orthant node. Apply the accuracy function. If it returns **TRUE**, or if the node is a leaf node, then inverse-transform the data, and extract and return the data that is within the subset's extent.
4. Recursively apply this procedure to all the children of the current orthant node, and return the union of the returned values.

Fig. 6. Subset algorithm.

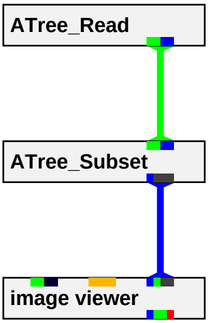

Fig. 7. Simple AVS network which displays a spatio-temporal subset of original raster data.

The performance of subsetting depends on what location code scheme is used since the bottleneck is disk/tape retrieval. Recall that the location code scheme defines how the data is clustered on the archival storage. Hence the location code scheme should be chosen according to the anticipated access pattern of datasets so that the subset operation is executed as efficiently as possible.

4.3 Value Search Algorithms

A *search object* needs to be specified as an input parameter of the content search operation. It is a generic representation of the feature that we are searching for. A content search algorithm returns the (hyper-rectangular) regions of an ATree that contain the search object. In addition to a search object, a *test function* needs to be specified. The test function takes a search object and an orthant node and returns one of: **NO**, meaning that the object is not present in the region corresponding to the orthant node; **REGION**, which represents a subextent of the orthant that contains the feature; and **MAYBE**, meaning that we need to go deeper in the search to obtain an answer.

5 User Interface

The ATree data structure is implemented as a C++ object library. Currently, the ATree uses a B-Tree as its underlying storage mechanism.

We have also implemented a suite of AVS [1] modules to operate on ATrees. AVS employs the boxes-and-arrows visual programing paradigm. Each AVS module represents some high level operation and is graphically represented by a box. Functionally, a module is a Unix process with a number of input and output arguments, called input and output ports. An output port of one module can be connected to an input port of another module, provided that the port types match. This connection is graphically represented by a colored line (arrow) between modules. The line represents data flowing from the first module into the second, and the color of the line represents the data type. Figure 7 is an example of a simple AVS network of modules.

Given the sizes of ATree data sets, it is not possible to pass a whole data set from one module to another. Rather, the following approach is taken. The ATree object library implements the ATree data structure as permanent objects (in the OO sense). Only a reference to a permanent ATree object is passed between two modules, so there is very little overhead in communication between modules.

We have implemented the following modules:

Build. Input to this module is a raw data set. The output from this module is the reference to the new ATree object. In addition, this module takes a number of parameters which specify how the resulting ATree is built. These parameters include Subdivision Scheme, Split Criterion, Transformation Function, and Location Code object specifications.

Subset. This module is a basic spatial query. The input to this module is a hyperrectangle, and the output is a data set corresponding to the specified spatial domain.
There are two versions of this module. The first version outputs another ATree as the result of this operation, and the second version outputs a multidimensional field in standard AVS format. This field can be then piped into any other AVS module that takes field data as input for further analysis and visualization.

Composite. This module is a spatial join operation (e.g., [14]). The input is two ATrees which are combined into a single data sets: $d_3(x) = f(d_1(x), d_2(x))$ where d_1 and d_2 are input data sets, d_3 is an output data set, $x \in$ domain(d_1) $\cup$ domain(d_2), and f is the composite function.
There are two versions of this module. The first version outputs another ATree. The second version outputs an AVS field.

Figure 8 shows the AVS network used to generate Figure 1.

6 Conclusion

Objectives of research in global change are to study, predict, and understand changes, both natural and anthropogenic, in the Earth system [6]. This requires improvements in observations, models, and information systems. Our research addresses improvements in information systems—that is, the development of a data structure to facilitate the construction of a database management system that efficiently indexes and accesses large data-sets.

To summarize, we believe that the ATree data structure meets the demands of interactive scientific applications used in global change research, i.e., data analysis and visualization. The ATree data structure will permit global change

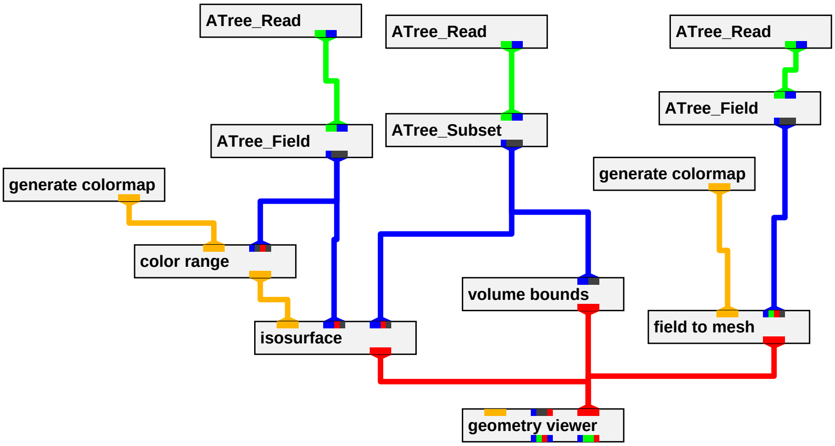

Fig. 8. AVS network used to generate a three-dimensional iso-surface shown in Figure 1.

scientists to store large amounts of data such as the data generated by satellite observations. In particular, the ATree data structure is suitable for storing abstracts as well as original data. It can store data at various resolutions. It intrinsically partitions data and organizes the chunks in a linear sequence onto secondary/tertiary storage. It allows quick access to a subset of interest, and is suitable for feature-based queries.

There are a number of directions for future research. They include comparisons with other methods and simulations of storing ATrees on tertiary storage. Note that a top-down build algorithm is useful for many compression algorithms. However, the drawback is a need to have all data accessible at once. Hence, it is worhtwhile to experiment with bottom-up and hybrid build algorithms. A bottom-up algorithm is a local process. It starts from smallest orthant nodes and merges the nodes as it moves up the tree to process larger nodes. A hybrid build algorithm is a combination of the top-down and bottom up build algorithms.

It is also worthwhile to investigate how to implement feature-based queries. Examples of such queries are: “Find all spatio-temporal regions that contain a hurricane” or “Find all spatio-temporal regions which exhibit the El-Niño pattern”. In addition, experiments with different compression techniques (e.g., DCT [11], wavelet transforms [10]) would be worthwhile.

References

1. Gaining insight through data visualization – the competitive advantage for data intensive applications — technology white paper. Technical report, Advanced Visual Systems, Inc., 1997.
2. W. G. Aref and H. Samet. Efficient window block retrieval in quadtree-based spatial databases. *GeoInformatica*, 1(1):59–91, April 1997.
3. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.
4. S. S. Dixit and Y. Feng. Hierarchical address vector quantization for image coding. *CVGIP—Graphical Models and Image Processing*, 1(53):63–70, January 1991.
5. J. A. Dutton. The EOS data and information system: concepts for design. *IEEE Transactions on Geoscience and Remote Sensing*, 27(2):109–116, 1990.
6. Committe on Earth and Environmental Sciences. *Our Changing Planet: The FY 1992 U.S. Global Change Research Program.* Office of Science and Technology Policy, Washington, D.C., 1991.
7. J. A. Fine. Abstracts: A latency-hiding technique for high-capacity mass-storage systems. Technical Report Sequoia 2000 #92/11, University of California, Berkeley, 1992.
8. G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing.* IBM Ltd., Ottawa, Canada, 1966.
9. J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proceedings of the Third ACM SIGACT–SIGMOD Symposium on Principles of Database Systems*, pages 181–190, Waterloo, Canada, April 1984.
10. W. H. Press. Wavelet transforms. *Harward-Smitsonian Center for Astrophysics*, (Preprint No. 3184), 1991.
11. K. R. Rao and P. Kip. *Discrete Cosine Transform—Algorithms, Advantages, Applications.* Academic Press, Inc., London, 1990.
12. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, MA, 1990.
13. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.
14. H. Samet and W. G. Aref. Spatial data models and query processing. In *Modern Database Systems, The Object Model, Interoperability and Beyond*, W. Kim, ed., pages 338–360. ACM Press and Addison-Wesley, New York, NY, 1995.
15. H. Shoshani. Properties of statistical and scientific databases. In *Statistical and Scientific Databases*, Z. Michalewicz, ed. Ellis Horwood, February 1991.
16. M. Stonebraker, J. Frew, and J. Dozier. The sequoia 2000 project. In *Proceedings of the 3rd International Symposium on Large Spatial Databases, SSD'93*, Singapore, June 1993.
17. S. L. Tanimoto. Image transmission with gross information first. *Computer Graphics and Image Processing*, 9(1):72–76, January 1979.