Scientific data models for large-scale applications

Lloyd A. Treinish
`lloydt@watson.ibm.com`

IBM Thomas J. Watson Research Center
Yorktown Heights, NY

Background

Obviously, data management is critical for appropriate and effective utilization of large volumes of data. In scientific and engineering computing problems today there are considerable independent efforts for data collection and simulation. To properly understand a phenomenon of interest, one much consider a notion of data *fusion*, by which these disparate sources can be utilized. For example, such data may come from

- remotely-sensed or in-situ observations in the earth and space sciences
- seismic sounding of the earth for petroleum geophysics (or similar signal processing endeavors in acoustics/oceanography, radio astronomy, nuclear magnetic resonance, synthetic aperture radar, etc.)
- large-scale supercomputer-based models in computational fluid dynamics (e.g., aerospace, meteorology, geophysics, astrophysics), quantum physics and chemistry, etc.
- medical (tomographic) imaging (e.g., CAT, PET, MRI)
- computational chemistry
- genetic sequence mapping
- intelligence gathering
- geographic mapping and cartography
- census, financial and other "statistical" data

The instrumentation technology behind these data generators is rapidly improving, typically much faster than the techniques available to manage and use the resultant data. In fact, an onslaught of orders of magnitude more data from these and other sources is expected over the next several years. Hence, greater cognizance of the impact of these data volumes is required. For example, NASA's Earth Observing System (Eos), which is planned for deployment by the end of the century, will have to receive, process and store up to ten TB (10^{13} bytes) of complex, interdisciplinary, multidimensional earth sciences data per day, for over a decade from a number of instruments. These data will be compared and utilized with models. Another example is the work under the US Department of Energy Accelerated Strategic Computing Initiative (ASCI) to shift from physical testing to computational-based methods for ensuring the safety, reliability and performance of the nuclear weapons stockpile. This requires the most advanced, state-of-the-art scientific simulations derived from a number of distinct codes. The results of such codes must be coupled and compared to the archived data from the physical testing. Given the fact that such data sets are or will be generated, what can be done to cope with this deluge from the perspective of their access, management and utilization?

The ability to generate pictorial representations of data is currently in vogue as being the answer. This concept of (scientific) data visualization really implies a method of computing that gives visual form to complex data using graphics and imaging technology. It is based upon the notion that the human visual system has an enormous capacity for receiving and interpreting data efficiently. Despite the advancement of visualization techniques for scientific data over the last several years, there are still significant problems in bringing today's hardware and software technology into the hands of the typical scientist. Some of these same problems do occur in more general processing and analysis of scientific data in many disciplines.

For example, data management is required to make such computing effective, which can be expressed by the need for a class of data models that is matched to the structure of scientific data as well how such data may be used. The critical component of data management is typically missing in many such computing efforts. When this concept is scaled to support large data sets (e.g., a few GB), several critical problems emerge in the access and use of such data, which brings the problem back to the data level. This circular reasoning does not imply that applications such as visualization are not relevant, but the challenge is more complex than simply flowing data from storage and looking at the pictures or even "pointing" back to the numbers behind the pictures. This requirement for scientific data models extends beyond the definition and support for well-defined physical formats. It must include the logical specification of self-documenting (including semantics) scientific data to be studied via a visualization system and data derived through the operation of the tools in such a system.

Recently, considerable attention has also been placed on metadata support for the management of current and past large data streams because through it a scientist would be able to select data of interest for analysis. However, if adequate mechanisms for use of the actual data that meet the requirements and expectations of the scientific users of such data are not available, then the efforts to generate and archive the data and supporting metadata management will be for nought. Although it is beyond the scope of this discussion, advances in visualization and data structures can also be applied to metadata management in the form of browsing, support of spatial search and selection criteria, etc.

What can be done? A place to begin is a discussion about solutions for how data should be organized, managed and accessed, which is often not adequately addressed in visualization and related software. The following is a survey of some of the methods and requirements for data management in visualization. It is hardly meant to be either exhaustive or definitive, but merely as an introduction to an important topic.

Data characteristics

One way of expressing the need for a data model is by defining what is meant by data. There are a tremendous number of sources of scientific data, be they computed or measured. Even from a single source there can be a wide variety of data sets. Each such data set typically contains several independent variables such as time, one or more spatial, spectral, etc. variables, and of course, many dependent variables, where the interesting science is stored. There can be a bewildering range of underlying formats, structures, arrangements and access methods for these data. To attempt to bring some simplifying order to this chaos, consider six key attributes of data: dimensionality, parameters, data type, rank, mesh structure, and aggregation.

Any data set may be considered as a single or multi-valued function of independent variable(s). These independent variables may be called dimensions. The number of independent variables may be called the dimensionality of the data. It is the fundamental characteristic of the data. Such dimensions may be space (length, width, height), time, energy, etc. For example, zero-dimensional data are just numbers such as sales, while two-dimensional data could depend on an area such as barometric pressure over a state. Some complex data may have five or more dimensions.

The function(s) composing a data set really are dependent variable(s) -- the data themselves, which may be called parameters. They are dependent on the dimensions, such as sales or temperature. Thus, data implies a parameter or field of one or more (dependent) values that is a function of one or more (independent) variables,

e.g., $[y_1, y_2, \ldots, y_m] = [f_1(x_1, x_2, \ldots, x_n)$ (1)

$$\begin{array}{l} f_2(x_1, x_2, \ldots, x_n) \\ . \\ . \\ . \\ f_m(x_1, x_2, \ldots, x_n)] \end{array}$$

These functions are continuous in nature, but sampled or discretized in a fashion often dictated by the specific computations to be performed. Operations imply a process of transformation between different functions of this class, whether it is solved as a set of partial differential equations that define flow of heat or generating pixels as a rendering of some geometry.

The data type includes the physical primitive, which describes how data values are stored on some medium (e.g., byte, int, float, etc.). It can include machine representations (e.g., little endian vs. big endian, IEEE vs. VAX, etc.). In addition, there can be a category of such types, i.e., real, complex or quaternion.

A parameter may have more than one value, which is characterized by tensor rank. Rank 0 is a scalar (one value), such as temperature (a magnitude -- a single-valued function). Rank 1 is a vector such as wind velocity (a magnitude and a direction: two values in two dimensions, three values in three dimensions). Vectors of size, n, are n-valued functions. Rank 2 is a tensor such as stress on an airframe (four values in two dimensions, nine values in three dimensions). A rank 2 tensor in n-dimensional space is a n x n matrix of functions (e.g., stress). Dimensionality and rank are thus, related. The number of elements in a particular parameter is dr, where d is the dimensionality and r is the rank. A distinction can be made between a 2-vector, 3-vector and two scalars in 2-space vs. a 2-vector, 3-vector and three scalars in 3-space independently of data type primitive or underlying mesh structure, if any. As with dimensionality, rank may be large for very complex data.

Such a taxonomy typically does not include those data directly associated with graphics, which is beyond the scope of this discussion. An important point to consider is that these characteristics should be considered more or less independently of each other to maximize flexibility in actual visualization software.

As with data, the techniques used to visualize may be similarly classified by their dimensionality. Because visualization implies creating a pictorial form for data, there is an implied geometrical relationship. Table 1 summarizes the dimensionality of typical visualization geometries.

Table 1. Dimensionality of Visualization Geometry

Dimensionality	**Geometry**
0	Point
1	Line
2	Polygon
3	Volume

Of course, a geometric primitive of lower dimensionality may be imbedded in a higher dimensional space such as a line on a plane or in a volume or a surface in a volume.

However, the dimensionality of visual representation may be different than that of the data, depending on the specific visualization task. In addition, these geometries may be used in different ways to create a specific visual representation.

Table 2 illustrates some of the diversity of visualization techniques that may be required in a system when only considering the earth and space sciences. Examples are listed with different dimensionalities and rank. Again, the number of elements in a particular quantity is d^r, where d is the dimensionality and r is the rank. The suite of techniques should accommodate time-dependent quantities as having another scalar dimension, which can be mapped into a specific visualization primitive or "axis." In a complementary fashion, animation should be treated as an additional scalar "axis" for sequencing of other visualization strategies in either a discrete or con-tinuous fashion, which may be hardware dependent. It should be noted that the example data types are not intended to be all encompassing and the specific visualization methods are not meant to be definitive mappings for specific data types nor exhaustive -- but only to impart the notion of the amount of diversity.

Table 2. Visualization Strategies for Data of Different Dimensionality and Rank

Dimension/ Rank	Example Data Types	Discrete Strategies	Continuous Strategies
0/0	point temperatures	histogram	
1/0	time histories altitude profiles zonal means	2d scatter plot 2d histogram	2d line plot
1/1	currents	arrows	
2/0	grid/image zonal profiles particle spectra	multiple 2d plots 3d scatter plot 2d pseudo-color scatter plot	iso-contours wire-frame surface shaded surface pseudo-color image
2/1	ocean surface currents	arrows	stream lines or ribbons
2/2	stress	pseudo-color arrows	pseudo-color stream lines or ribbons

3/0	grids/images gridded profiles zonal profiles	3d pseudo-color scatter plot sliced isosurfaces transparent isosurfaces "solid" contours	pseudo-color shaded surface 3d iso-contours volume rendering
3/1	winds	"solid" arrows	stream lines or ribbons

Data mesh structure

Perhaps the class of characteristics for which there is the most confusion is in the specification of mesh or grid types because of the typical use of domain-specific terminology. In general, a mesh describes the base geometry for the mapping of the functions or dependent variables to some (physical) coordinate system. Such a mesh may be explicitly or implicitly positioned. In the latter case, it is preferred to store the positioning information implicitly. Some implementations may store it explicitly. In most cases there is a topological relationship or cell primitive connecting these positions. Table 3 summarizes the dimensionality of typical mesh cell primitives.

Table 3. Dimensionality and Mesh Cell Primitives

Dimensionality	Primitive
0	not applicable
1	line
2	triangle quadrilateral
3	tetrahedron hexahedron prism pyramid

The following are a few cases of commonly used mesh structures:

1. Regular grid with regular positions and regular connectivity.

In this case, the two-dimensional primitive is a rectangle while the three-dimensional primitive is a parallelpiped. Usually, this case would imply cartesian coordinates. The mesh may be implicitly stored in a compact fashion via a specification of an ordered pair for each dimension (i.e., the origin and spacing in some coordinate system). The entire mesh may be formed by taking a product of the specifications across each dimension. An example would be a grid of temperatures over a rectilinear map of the earth's surface.

2. Deformed regular or curvilinear or structured grid with irregular positions and regular connectivity.

In this case the primitives are the same as in case 1. Non-cartesian coordinates may be implicitly supported through such a deformed structure. Generally, the mesh is explicitly stored via a specification of a node position along each dimension (axis) in some coordinate system. The entire mesh may be formed by taking a product of the specifications across each dimension. An example would be a grid of pressures over a an airframe.

A variation in this case and in case 1 would be for a partially regular grid, which has one or more dimensions being regular (i.e., case 1.) and the balance being irregular.

3. Irregular "regular" or structured grid with irregular positions and regular connectivity.

In this case the primitives are the same as in case 1. and 2. but with irregular sizing and spacing, which may include holes. Positions are specified explicitly. Sparse matrices, grids or meshes can also be considered in this category. However, indexing schemes can be introduced to eliminate wasted storage. An example would be several satellite images with gaps in coverage.

4. Unstructured or irregular grid with regular or irregular connectivity.

In this case, the two-dimensional primitive is typically a triangle while the three-dimensional primitive is typically a tetrahedron. However, it may be prismatic, icosahedral, hexahedral, etc. or even variable. Positions of each node are specified explicitly while a connections list identifying the relationship and order of each node on each primitive is generally required. An example would be a finite element mesh of the structural integrity of the frame of a car.

5. No grid with irregular positions and no connectivity.

In this case the data are scattered or available on specific points, which are explicitly identified. An example would be measured rainfall in specific towns.

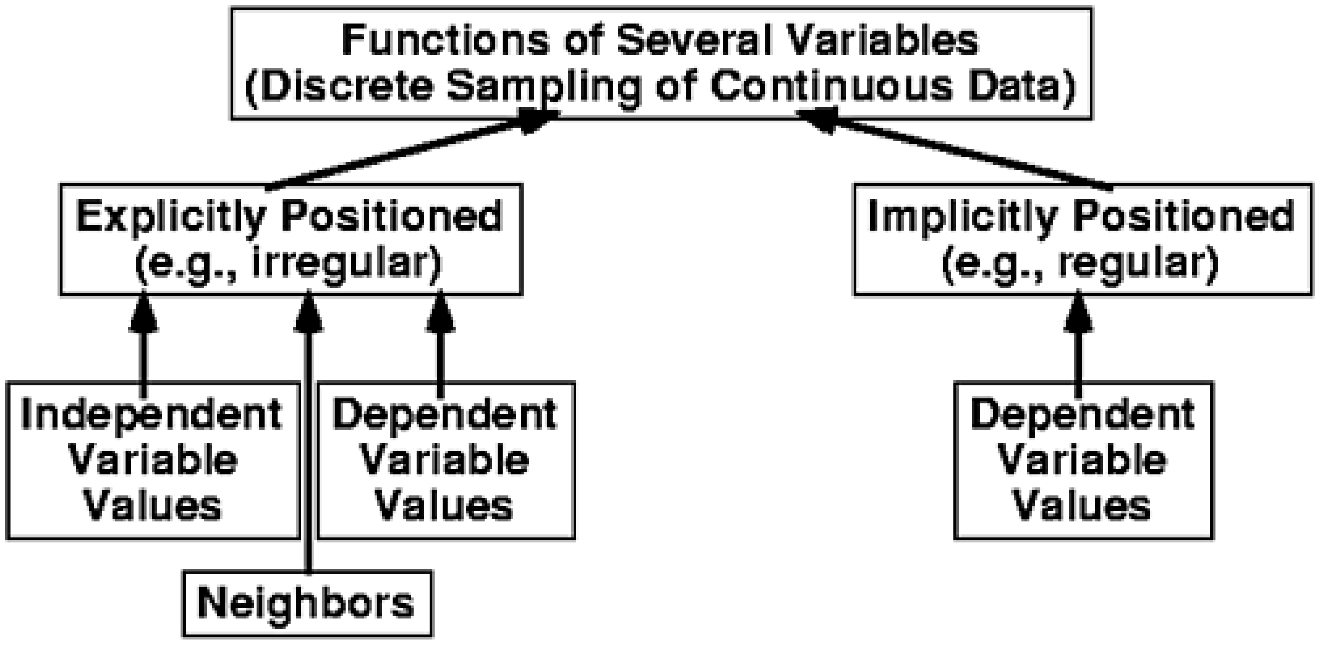

Functions of Several Variables
(Discrete Sampling of Continuous Data)
Explicitly Positioned
(e.g., irregular)
Implicitly Positioned
(e.g., regular)
Independent Variable Values
Dependent Variable Values
Neighbors
Dependent Variable Values

Figure 1. Example Taxonomy of Data Properties and Meshes

The notion of taking a product to form a mesh from vectors of positions can be applied to connections between positions. For example, regular connections of n points is defined by a set of n-1 line segments. A product of two sets of connections is a set of points obtained by summing one point from each of the terms in all possible combinations. Hence, a product of two regular connections composed of line segments would be a set of squares. A product of a regular connections of line segments and an irregular one of triangles would be a partially regular set of prisms. Figure 1 shows a potential taxonomy for these different mesh classes with respect to the functional description in equation (1).

Some of the two-dimensional mesh structures are shown in Figure 2. Figure 3 show similar mesh structures, but in three dimensions. However, conventions are required for the specification of where the data (function is applied) with respect to the connection elements or topological cell primitives. The data may apply to each node or position of a grid. Alternatively, the data may apply to a cell center or face, or even an edge. Although many cases usually apply only one convention for an entire mesh, it is not always true. Some finite element meshes may use more than one type of primitive and convention.

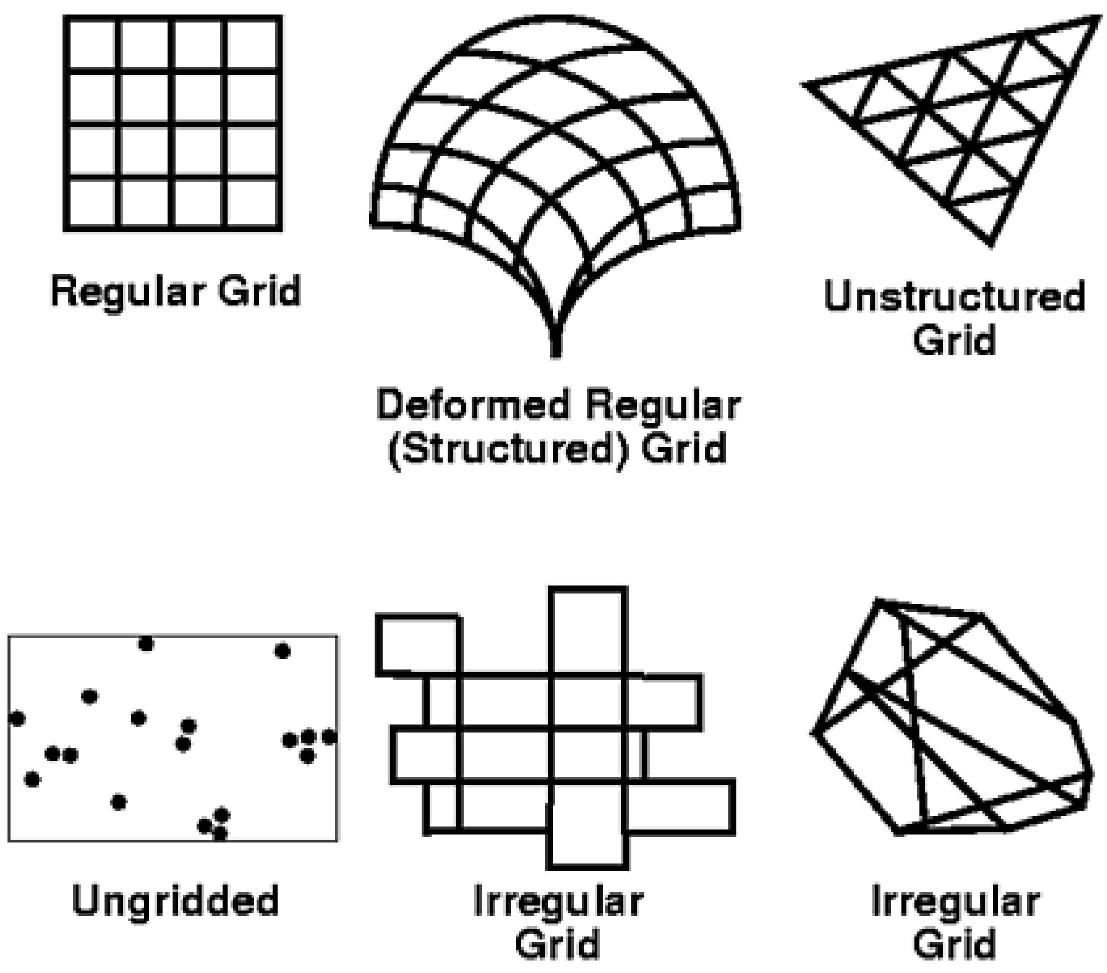


Figure 2. Example Two-Dimensional Mesh Types

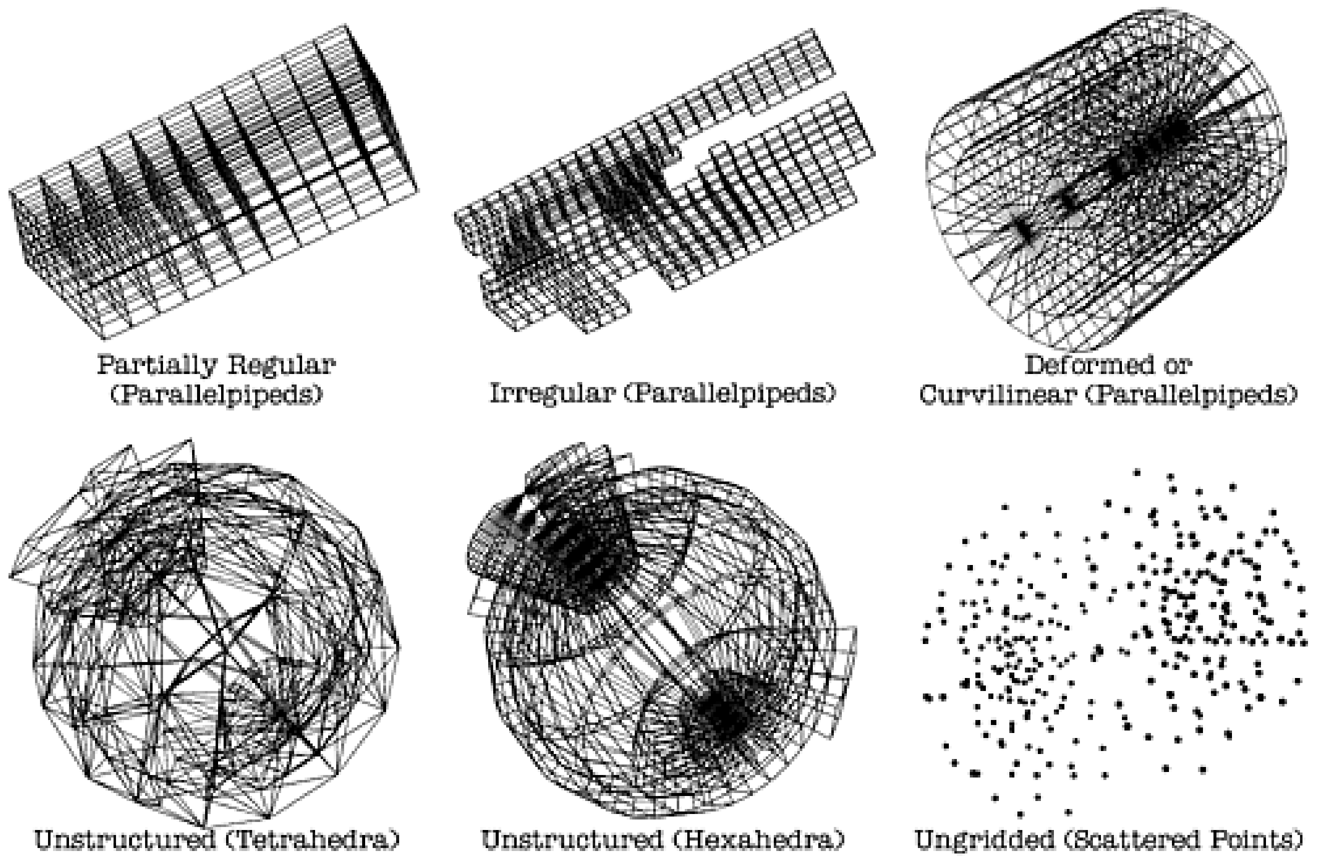

Partially Regular (Parallelpipeds)
Irregular (Parallelpipeds)
Deformed or Curvilinear (Parallelpipeds)
Unstructured (Tetrahedra)
Unstructured (Hexahedra)
Ungridded (Scattered Points)

Figure 3. Example Three-Dimensional Mesh Types

Data aggregates

Often there is a need to form aggregates of data. In dependent-variable space, one could have a collection of parameters or fields over the same or different grids that could be treated as single entity. In that sense, an aggregate or group could be composed of members that are either a single field or another group. This mechanism can be used to define simple tree structures. In independent-variable space, one could have a collection of meshes. For example, in aerospace fluid dynamics simulations, a computation is often performed over several intersecting grids. Such a multigrid solution permits the definition of variable grid resolution and regularity around airframe structures such as an engine nacelle or a wing. In this case, one could treat the collection of meshes as a single entity, although a mechanism must be defined for accommodating regions of invalidity where grids may intersect. A similar problem occurs with observational data, where there may be no data for some grid nodes. Another type of mesh aggregate can result from a hybrid collection of meshes of different cell primitives, where within each mesh, the cell is the same.

A special case of aggregates can be called series, where the tree structure has only one level of children. The classic example is a time series, where there is multiple instances of some field or aggregate over a constant or changing mesh. Generically, such a series does not have to depend on time, but could apply to any sequencing of events.

Temporal data

The support of temporal data as time series is an interesting matter. How should time be handled? There are two flavors of temporal support: as a running clock and epoch-based. In either case, time tags could be viewed like an additional dimension in the position component of a field with "linear" connectivity. One could then use a compact representation for constant delta t. Otherwise, each time tag (position) could be specified individually for irregular spaced time steps. Obviously, the other method of supporting time tags would be associating each tag with a member of a (time) series.

The running clock approach supports elapsed time from some base (i.e., such as the origin of a position component dimension) like time of launch or start of simulation. The simplest implementation would be the use of a floating point representation of linear time in user-defined units (e.g., milliseconds, seconds, minutes, hours, days, years). Optionally, the elapsed time could be in a clock or calendar representation as a float (e.g., HHMMSS.s, YYDDD.d, where HH = hour [0-23], MM = minute [0-59], SS = second [0-59], s = fractional second, YY = year, DDD = day of year [0-365 or 366], and d= fractional day).

Epoch-based time tags represent an "absolute" standard time from a standard reference epoch in a linear, clock or calendar representation. Such tags are more common with observational than simulated data since the former often require a link to the real world as well as registration with other observations. Generally, GMT is used as the standard time frame (at least for the earth). Example reference epoches often start exactly at midnight on January 1 of some specific year (e.g., 0, 1900, 1950) according to an accepted (i.e., European) calendar. Some example conventions for representation include the following:

- milliseconds since midnight, 1 January 0 AD (double)
- year/month/day since midnight, 1 January 0 AD: YYYYMMDD (long)
- time of day: HHMMSS (long)
- year/month/day since midnight, 1 January 1900: YYMMDD.d (float)
- year/day of year since midnight, 1 January 1900: YYDDD.dd (float)
- year/month/day/hour/minute/second/msec since midnight, 1 January 0 AD: YYYYMMDDHHMMSSmmm (string)

Obviously, one can generate any number of acceptable and useful conventions ad nauseam. However, in almost all such cases there are significant problems in providing a 32-bit floating point representation with sufficient precision to handle both clock and calendar data together. A recent, related, and mostly commerical, example of this is the so-called Year 2000 problem, which resulted from storing a year as a two-digit integer since 1900. In addition, for whatever conventions are established, efficient algorithms for calendaric and temporal manipulation will be required (e.g., unit and scale conversion, leap year calculations) to keep these conventions reasonably transparent.

Implementations and techniques

Within the context of scientific computation there is a need for a class of data models that is matched to the structure of the data as well as how such data may be used. Traditional methods of handling scientific data such as flat sequential files are generally inefficient in storage, access or ease-of-use and prohibit effective data exchange and interoperability between applications, especially for large complex data sets, and input/output and floating-point-intensive applications like signal processing (e.g., inverse problems) and visualization. Modern, commercial relational data management systems do not offer an effective solution because they are more oriented to business applications. The relational model does not accommodate multidimensional, irregular or hierarchical structures often found in scientific data sets nor the type of access that associated computations require. In addition, relational systems do not provide sufficient performance for the size, complexity and type of access dictated by current and future data sets and their potential usage.

In contrast, these data base management systems have been quite viable for a class of non-spatial metadata management (e.g., warehousing, characteristics, etc.). Hence, current data base systems are not yet up to the challenge of supporting direct access to very large data sets.

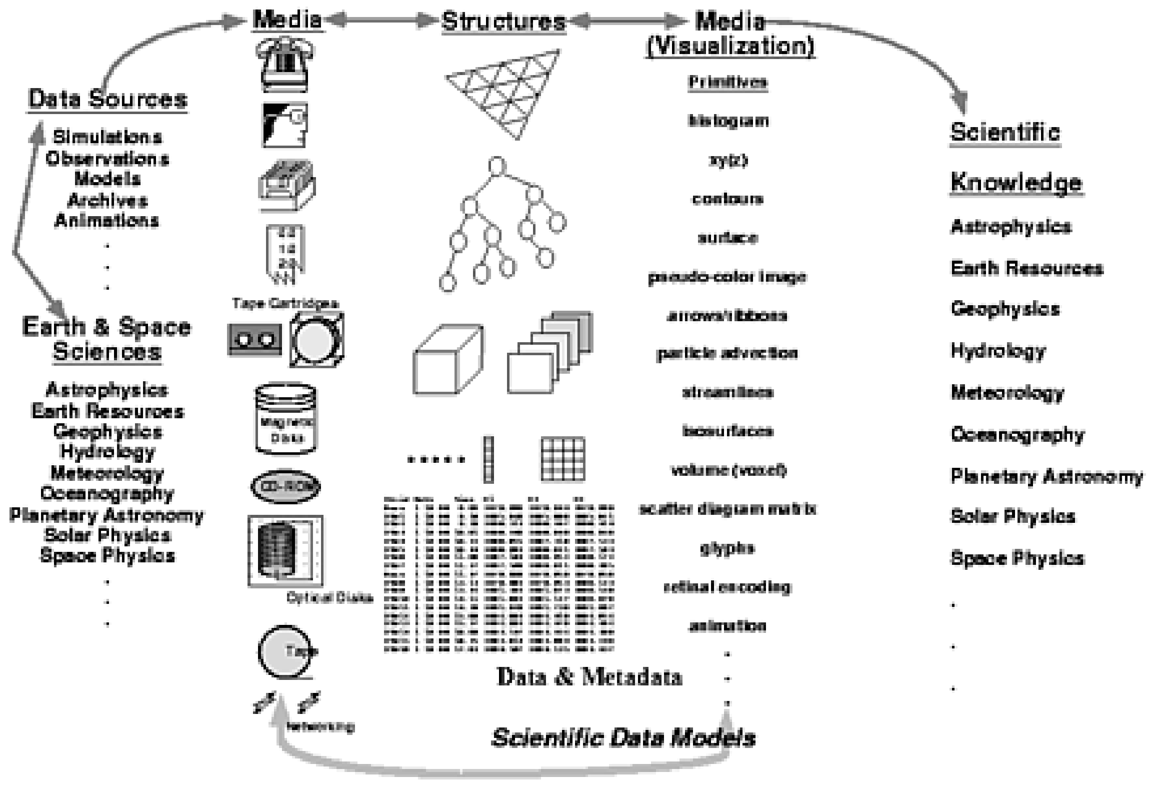


Figure 4. Data Management Layer for Visualization

Another way to view this idea is a layer that provides a logical link between the concepts that scientists use to think about their field (e.g., particle trajectory, cerebral cortex shape, plasma temperature profile, or gene) and the underlying data from simulation experiment or the storage system. In particular, this layer provides tools that are common to all applications for data definition, metadata support, and query formulation and execution. This layer supports computational, analysis and visualization tools, and provides the infrastructure for data representation and access (e.g., data model definition, metadata support, query formulation, etc.). Like a data base management system, it would reside above the operating system and enable the building of applications. Hence, scientific data models have the potential to hide the complexity of underlying computational systems for simulation, analysis and visualization, freeing scientists to focus on data comprehension by providing a common mechanism for access, utilization and interchange. The implementation of such data models, the management of and access to the data, should be decoupled from the actual visualization software. Conceptually, Figure 4 illustrates this notion, where the data model serves as a bridge between visual and data media.

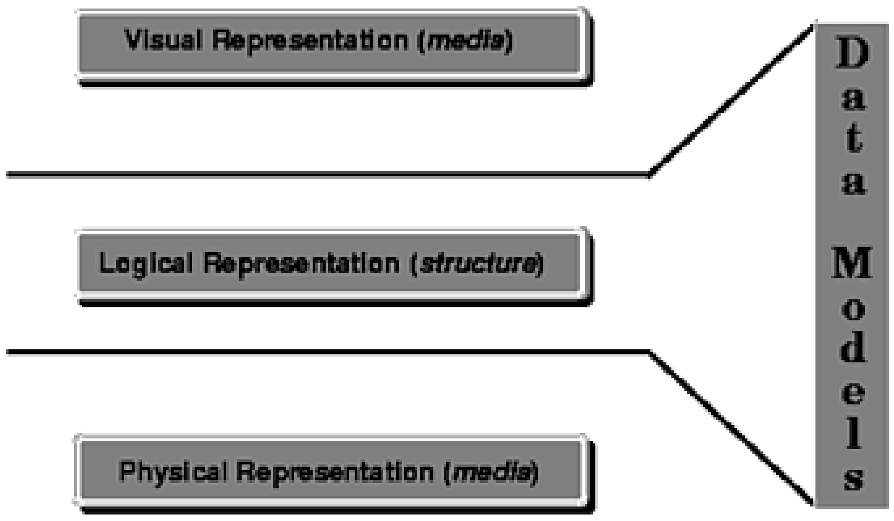

Visual Representation (*media*)
Logical Representation (*structure*)
Physical Representation (*media*)
Data Models

Figure 5. Role of Data Models in Visualization Software

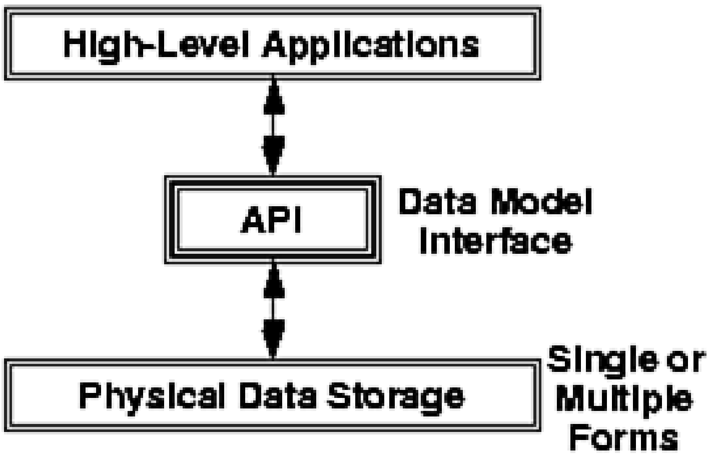
Therefore, there is a need for some type of data (base) model(s) that possesses elements of a modern data base management system but is oriented toward scientific data sets and applications. This intermediate approach should be easy to use, support large disk-based (perhaps other media as well) data sets and accommodate multiple scientific data structures in a uniform fashion. In the process of providing simple access to self-describing data, such a mechanism should match applications requirements for visualization as well as data analysis and management, and be independent of any specific discipline or source or visualization technique. Hence, data management as embodied as a data model(s) is as important a component of a data visualization system as underlying graphics and imaging technology. In other words -- borrowing from the data base community, there exists a physical representation (media), a logical representation (structures) and a visual media, more simply illustrated in Figure 5.

Many software packages touted as supporting visualization ignore the issue of data management. Access to data is often provided via a simple flat file structure that may or may not be proprietary or left as an exercise to the user. Systems that support different classes of grids (e.g., regular/structured and unstructured) typically do not provide uniform mechanisms for their use. In any of these cases, the extensibility of such software to many, potentially very large data sets for disparate applications is extremely limited. Any process that accesses data implies the development of software that can manage arbitrary data sets and possesses different tools for displaying (or working with) data. There must be a clean interface between the data and the display of the data, so that arbitrary data can be accessed by the visualization software. As a consequence of such an approach, a software system of this design has an open framework. It can ingest arbitrary data objects for visualization, and other visualization techniques can be added independent of the application. This implies that a significant reduction in long-term software development costs can be realized because new data sets do not require new display software, and new display techniques do not require new data access software. A recommended initial approach is to attempt to characterize and categorize the data in the domain of interest according to the taxonomy outlined in previous section.

Implementation genre

These classic limitations have been recognized by a few groups in the support of a number of scientific applications. As a result, several data models have been defined, some within very domain-specific contexts while others being more general. Recently, a few of these models have been associated with software, including visualization and have associated language bindings that represent the implementation of one or more abstract data types. These implementations can be characterized in the following manner:

- Unified -- utilizes a single, generalized data model
- Diffuse -- utilizes multiple data models
- Focused -- utilizes a single, domain-specific data model



A unified implementation includes an Applications Programming Interface (API) and/or language bindings, which defines operations to be done with data. This implementation is shown schematically in Figure 6. Such an interface may be "object-oriented", where such operations imply methods that are associated with a class of objects. However, the notion of an abstract data type is sufficient for such an implementation.

Figure 6. Unified Implementation (Interface Defines Operations)

Hence, the API hides the details of the physical form(s) of storage for data. In general, such implementations permit easy incorporation of new data and applications and offer the potential to develop highly optimized performance for generic applications.

The primary limitation of this approach relates to the data model itself. Such implementations are usually hard to extend for new classes of data not considered in the original data model definition. A diffuse implementation includes multiple APIs, which define underlying data structures. Each of these structures and hence, APIs, are associated with a single physical form of storage. This implementation is shown schematically in Figure 7.

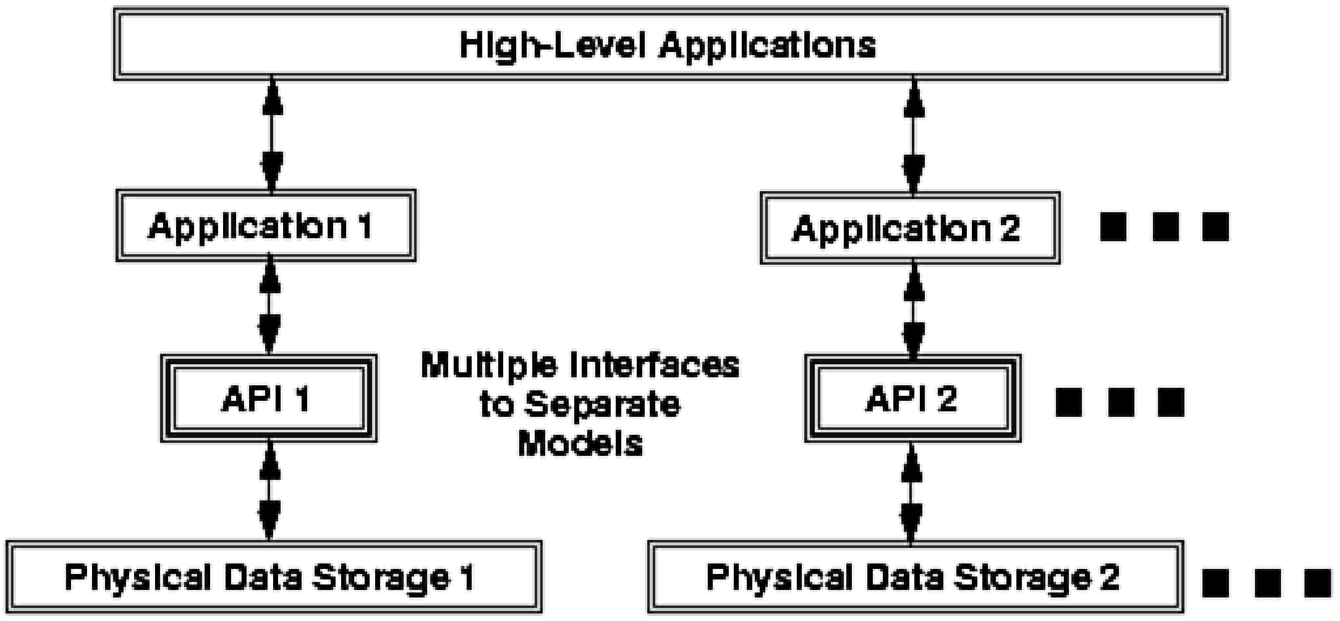
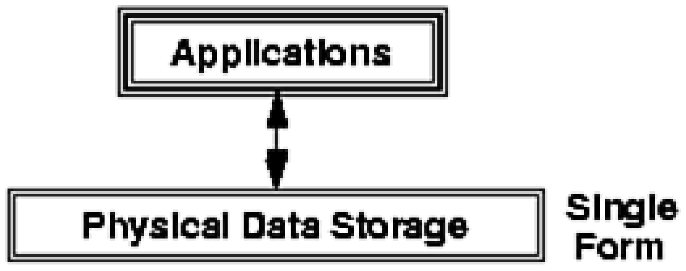


Figure 7. Diffuse Implementation (Interface Defines Structures)

Usually, separate implementations of specific applications are required for each interface, which may be integrated in some fashion by a higher-level application. Such implementations permit easy incorporation of new data because a new application, interface and structure is added. Hence, it becomes difficult to operate with more than one of the types of data simultaneously.

As a result, these implementations are usually difficult to optimize or extend for generalized applications.



A focused implementation usually does not include an API for the data model. This implementation is shown schematically in Figure 8. Any interface to data in such a system defines the underlying physical data format.

Figure 8. Focused Implementation (Interface Defines Formats)

Therefore, it becomes easy to optimize the performance for very specialized applications and quite difficult to extend to other data and applications.

To illustrate this taxonomy, a few examples of each type of implementation for data management or applications enabling and visualization are discussed briefly below. These examples are either public domain or commercial software that are used in the scientific community. The list is meant to be illustrative, not exhaustive or comprehensive. Additional details concerning these implementations are given in the final section of this paper.

Implementations for data management or applications enabling

Common Data Format (CDF), developed at NASA/Goddard Space Flight Center, was one of the first implementations of a scientific data model. It is based upon the concept of providing abstract support for a class of scientific data that can be described by a multidimensional block structure. Although all data do not fit within this framework, a large variety of scientific data do. From the CDF effort spawned the Unidata Program Center's netCDF, which is more focused on issues of uniform data transport. Hence, CDF and netCDF can individually be characterized as being distinct unified implementations.

Both netCDF and CDF support the same data model -- the idea of a data abstraction for supporting multi-dimensional blocks of data -- since netCDF is a separate and more recent implementation of the ideas that were developed in the original VAX/VMS FORTRAN version of CDF in the mid-1980s. Although the model is the same, the interfaces are quite different. The current release of CDF is much newer than that of netCDF, where the latter has only had incremental improvements and new ports since its initial implementation. These systems are extensible by the user, and conventions have been established in some organizations to ensure proper interpretation when data are exchanged.

NetCDF has only one physical form -- a single file written according to the IEEE standard via the eXternal Data Representation (XDR) protocol developed by Sun Microsystems and placed in the public domain. The current implementation (3.3) retains this physical format, by now bypasses the XDR software layer for efficiency. The multi-dimensional arrays are written by C convention (last dimension varies fastest). This implementation has proven to be very convenient and easy to use. As a result, the software has been ported to a number of platforms, is widely used in the atmospheric sciences and other research communities and can be employed with some independent visualization software. However, this approach does have its performance limits when scaled to large data sets, used by FORTRAN programmers or operated on non-IEEE machines. Many operating systems have relatively small limits (either actual or practical) on the size of physical files, which would require an user to implement a multiple file solution. In addition, the current software supports only limited direct editing or other transactions on the files in place (i.e., without copying). Currently, Unidata has only provided limited generic utilities for data in netCDF (e.g., netCDF operators) but many tools are available from third party sources (public domain and commercial).

In contrast, CDF supports multiple physical forms: XDR or native, single or multiple file (one header file and one file for each variable), row (i.e., C) or column (i.e., FORTRAN) major organization and the ability to interoperate among them, which includes compatibility with the original VMS FORTRAN implementation. This implies that for performance-critical applications, the appropriate physical form can be chosen. For example, where data portability is not critical and absolute performance is of greater importance (e.g., VMS), the support of a native physical format is critical. The CDF software has been ported to to a number of platforms and is most often used in the space physics and climate research communities. The CDF software supports caching and direct utilization of the file system to provide rapid access and in-place updates. Data in CDF are supported by a large and growing collection of both utilities and sophisticated general-purpose applications (some portable and some VMS-specific).

Another important effort has been the Hierarchical Data Format (HDF) developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign. This activity evolved from the need to move files of scientific data among heterogeneous machines, which grew out of the requirement to look at images and other data on personal computers, workstations, etc. that were generated on a supercomputer. HDF, which is also self-describing, uses an extensible tagged file organization to provide access to basic data types like a raster image (and an associated palette), a multidimensional block, etc. In this sense, HDF provides access (i.e., via its C and FORTRAN bindings) to a number (six) of different flat file organizations. Hence, HDF can be characterized as being a diffuse implementation.

The HDF software has been ported to a wide variety of platforms from personal computers to supercomputers. Currently, all of HDF's data structures are memory resident. Thus, the aforementioned discussion about the relative physical organizations of CDF and netCDF and performance tradeoffs do not apply to HDF, because that level of functionality is not provided. This limits the ability of an application using HDF software to effectively utilize or even randomly access large disk-resident data sets. The myriad of data types that HDF supports essentially through separate interfaces is through central registration at NCSA in contrast to the CDF/netCDF approach. There are obvious advantages and disadvantages of leaving full authority with the user or with the implementer. A particular choice of approach is highly application dependent. Thus, HDF is extensible to support additional models and formats but only at NCSA (e.g., recent efforts to support more complex data structures). HDF has been an extremely successful vehicle for creating portable data sets and driving a number of popular and visualization packages on a variety of platforms (Apple Macintosh, Windows and Unix), which offers further evidence as to the importance of codified data access techniques. Third-party applications have also been developed by commercial and academic groups to utilize data stored as HDF.

HDF has been extended with a new storage scheme called Vset, that is compatible with the extant HDF structures. It attempts to supersede some of the limitations inherent in the CDF/netCDF and the conventional HDF data models. It supports regular and irregular data and the ability to form hierarchical groupings. Another extension to HDF currently in development is to embody the netCDF and model and interface as an additional HDF object. This idea may be extended to the CDF model and interface as well. On-going work in this regard may enable HDF to provide an underlying array access layer that could be used by netCDF and HDF for various higher-order data types.

The Flexible Image Transport System (FITS) developed by the National Radio Astronomy Observatory is the standard interchange mechanism for astronomical (except for planetary) imagery and related data. It is well defined, self-describing but has no official software interface or language bindings. Hence, portability is implied at the physical byte level. Hence, FITS can be characterized as being a focused implementation.

Most databases for spherically distributed (e.g., astronomical, cartographic, climatological) data are not structured in a manner consistent with their geometry. As a result, such databases possess undesirable artifacts, including the introduction of "tears" in the data when they are mapped onto a flat file system. Furthermore, it is difficult to make queries about the topological relationship among the data components (e.g., adjacency of connected regions) using simple and purely symbolic means, that is, without performing real arithmetic. For example, hierarchical data structures are appropriate to support spatial data. A potential solution to these problems has been developed at NASA/Goddard Space Flight Center by applying recursive subdivision of spherical triangles obtained by projecting the faces of an icosahedron onto a sphere. The collection of these spherical triangles are managed with quadtrees. These sphere quadtrees are insensitive to the familiar distortions suffered in planar representations far away from the equator. Such distortions arise from the need to properly project data (i.e., flatten the earth) by preserving shape or distance, for example, in two dimensional visualizations. This data structure allows the representation of data at multiple resolutions and arbitrary levels and is supported with efficient searching mechanisms. It provides the ability to correlate geographic data by providing a consistent reference among data sets of different resolutions or data that are not geographically registered. The implementation of this idea represents a hybrid of the aforementioned approaches because it is very domain-specific, but possesses a welldefined data structure and interface as well as operations.

Implementations for visualization

The Visual Technologies Department at IBM Thomas J. Watson Research Center has developed a more comprehensive data model that includes curvilinear and irregular meshes and hierarchies (e.g., trees, series, composites), vector and tensor data, etc. in addition to the class of scalar, multi-dimensional blocks supported by the aforementioned implementations. Currently, the physical disk-based format, called dx, is complete and has been published, but it provides only simple sequential access. A portable, application-independent software interface for it is not yet available, and hence support for disk-based structures and database-type updates in place are not supported. The IBM Visualization Data Explorer software, developed by this same group, utilizes this format as one import/export mechanism. It is an extended, client-server data-flow system for general visualization applications. More importantly, however, is the implementation of the data model in this software package via an object-oriented approach to management of data in memory for computation. All operations on data within this software, independent of a role in generating pictures, work with shared data structures in memory via an uniform interface. The model utilizes the fact that researchers at Sandia National Laboratories observed that the mathematical notion of fiber bundles provides a useful abstraction for scientific data management and applications. The fiber bundle model is illustrated in Figure 9.

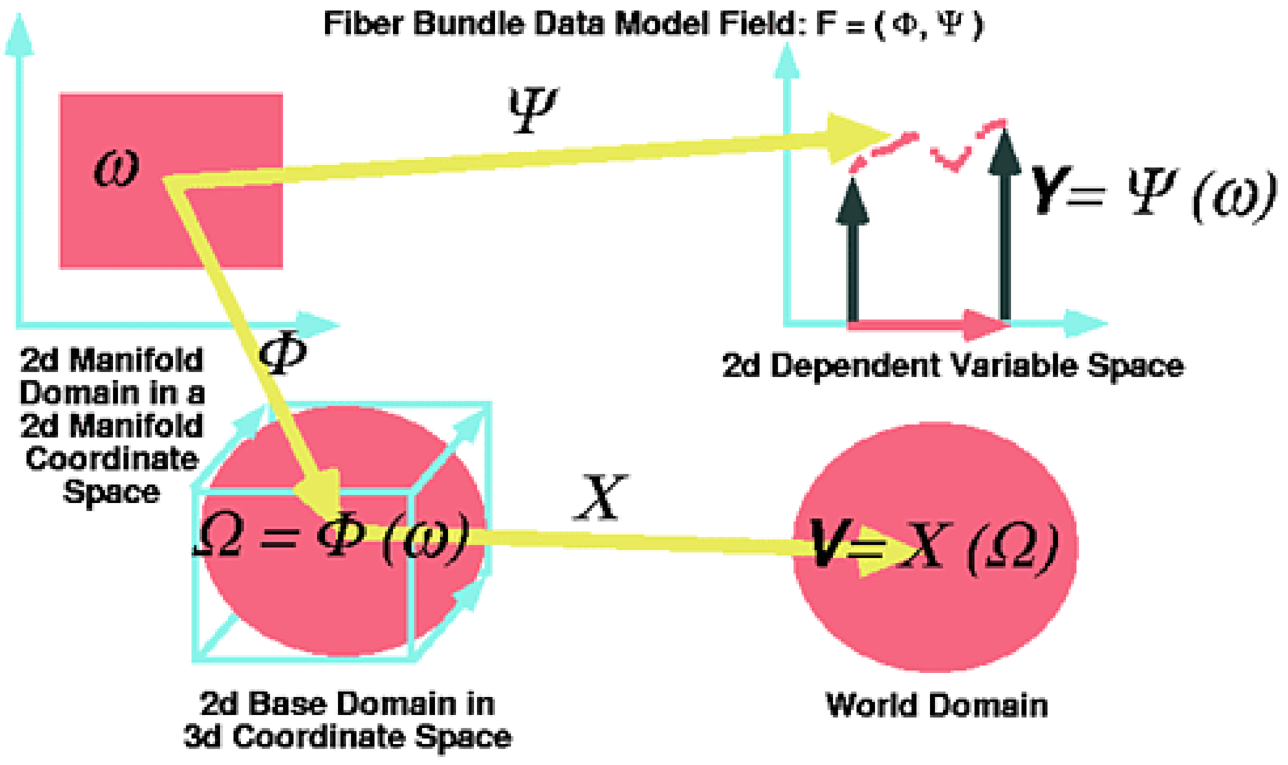

Fiber Bundle Data Model Field: F = (Φ, Ψ)
Ψ
ω
Y= Ψ (ω)
Φ
2d Manifold Domain in a 2d Manifold Coordinate Space
2d Dependent Variable Space
X
Ω = Φ (ω)
V= X (Ω)
2d Base Domain in 3d Coordinate Space
World Domain

Figure 9. The Fiber Bundle Field Data Model

In Data Explorer, this idea is specialized and extended to incorporate localized, piecewise field descriptions. This permits the same consistent access to data independent of its underlying grid, type or hierarchical structure(s) via a uniform abstraction. Data communication among subsequent operations is accomplished by passing pointers. In addition, sharing of these structures among such operations is supported. For example, when an image is rendered, or for that matter any computation in this system, the data structures can be linked (i.e., hierarchically) so that a path from the data to the pictures and back can be identified. Hence, IBM Visualization Data Explorer can be characterized as being an unified implementation.

The Application Visualization System (AVS) from Advanced Visual Systems, Incorporated and IRIS Explorer or simply Explorer from Silicon Graphics Incorporated are both distributed data-flow software packages for general visualization applications. From a data management perspective, both systems are conceptually quite similar. They incorporate separate underlying interfaces for different classes of data (e.g., structured vs. unstructured). As a result they often provide separate applications of similar functionality for different data (i.e., modules). Both packages also provide higher-level applications that integrate some of this functionality. Hence, AVS and Explorer can individually be characterized as being distinct diffuse implementations.

The Flow Analysis Software Toolkit (FAST) developed by Sterling Federal Systems, Incorporated for NASA/Ames Research Center is an integrated collection of tools designed for the visualization and analysis of the results of computational fluid dynamics (CFD) simulations. The performance of the tools and their interface are optimized toward researchers working on CFD problems. Hence, FAST can be characterized as being a focused implementation. More recently, researchers there have also utilized the idea of a fiber bundle to serve as an abstraction for CFD data to be visualized, and hence simplify the manipulation of large data sets. Their system, called SuperGlue, uses an interpretative language, based upon C and Scheme (a dialect of Lisp) for rapid prototyping of complex visualizations and user interfaces while effectively supporting code reuse. Therefore, SuperGlue can be considered a hybrid implementation because it uses a unified approach for handling data but within a very specific domain.

Metadata

Inherent in data models for visualization should be the ability to support self-describing scientific data structures. Generically, such self-descriptions are loosely referred to as metadata or information/data about data. Recently, considerable attention has been placed on metadata support for the management of current and past large data streams because through it a scientist would be able to select data of interest for analysis. For example, the National Space Science Data Center (NSSDC) at NASA's Goddard Space Flight Center among others has been instrumental in the implementation of several systems that provide metadata management services to the earth and space science research communities. Such support has primarily been for characteristic (i.e., directories and catalogs), physical inventory and temporal information with more recent efforts looking at problems in spatial (e.g., geographic) information. Unfortunately, most efforts at metadata management, including many of those within NASA have been divorced from the data themselves. An exception to this was the development of an information/analysis system for atmospheric research in the early 1980s. An attempt to bridge this gap between metadata management and data was one of the factors that lead to the initial development of the CDF, from which spawned (for some similar reasons) netCDF. It is in such a context that this discussion about metadata -- that is for the data themselves, what information is required to make the data sufficiently scientifically self-descriptive for "visualization."

In addition to self-describing data structures with well-defined software interfaces such as CDF, netCDF, and HDF that support metadata at some level, there are a number of more traditional approaches bundling useful scientific information with data. Typically this has implied sequential (i.e., magnetic tape) formats whose data contents are preceded by formatted text headers with little or no associated software. Among the better examples are ones that have been developed in the earth sciences in Europe (e.g., GF3, BUFR/GRIB) and the astronomical imaging community in the US (e.g., FITS). Recently, on-line versions of such formats have appeared as well as reading/writing FORTRAN programs and interesting applications. Although these formats were originally designed for the convenience of data producers trying to create magnetic tapes rather than more challenging applications like visualization, their developers and subsequent users have developed a rich lexicon of metadata that should not be ignored.

Data base management systems (DBMS) provide a capability to find, sort, merge, organize, update, and output diverse data types. However, most DBMSs have been designed primarily for archiving and managing data for a specific domain by developers with a background in computer science or related fields rather than a (physical) science discipline. The result is that these systems suffer from the intrinsic flaw of not effectively providing the capabilities needed by a casual or new user in that scientific discipline. These databases restrict the capabilities for managing the syntax of a domain as part of its data structure, have limited data structures which cannot represent explicit relationships between data classes and demand precise, mathematical query formulation for database interactions (i.e., SQL). In addition, such DBMS often exclude many of the data objects used in the scientific domain and do not efficiently store, index, or retrieve (i.e., in relational systems) image or spatial data. Hence, such data bases are difficult to design for scientific data and the users of existing database systems require an indepth understanding of the database architecture, data content, location, and query language in order to use the data effectively. When data structures exist in a large database (i.e., schemata), this problem is exacerbated, often making the database unusable in an operational environment. These limitations can apply for large scientific metadata bases as well as for the data themselves.

Classes of metadata

To help in the evaluation of what metadata and how it should be supported, the notion of metadata must be more clearly specified. Hence, four classes of metadata are defined: (1) what is needed to access the data; (2) what information is associated with or describes the data; (3) what data are associated with or define the data; and (4) what other documentation categorizes the data.

"Data Base" metadata

The first class -- what is needed to access the data -- is essentially what is refered to as metadata in the data base sense. Generically this metadata class includes data type primitives (e.g., byte, short, long, float, double, string) and structural information for each enumerated data object. The latter can include dimensionality (size, shape, [in]dependency), rank, positions, connections, and any referencing or indexing to other objects. For group objects, global information on the type of group would be necessary while similar information on individual members would by definition already be supported.

Attribute metadata

The second class of metadata is various types of information that is associated with data, generally at the level of an individual data object. Consistent with the CDF/netCDF parlance this class is called attributes. The attribute class of metadata should be extensible by the user. In other words, the user is free to define new attributes, even though some application software should not be expected to operate on such user-specific information except to acknowledge its existence and be able to "list" it. A discussion of individual examples of such metadata can be seen in the user's manuals for CDF and netCDF.

Ancillary (meta)data

In addition, to information about data there may be other data associated with data that are required for a complete definition. Such (meta)data is clearly numeric and could typically include spatial, temporal and spectral tags or locations associated with data, which can be accommodated via mesh specification.

Other (documentation) metadata

The last class of metadata helps the user to identify qualitative facts about the data. Such information is typically textual and probably is best supported by free text. There are several types of documentation in this category, which include laboratory notes or logs to identify what was seen, to identify what was done, etc. Another category would include a specification of what should be done with the data by the user or someone else. A third variety would include user and data set information to identify saved images or hardcopy, place data in the proper context, etc. These metadata are textual and could be supported in the same manner as text/string attributes. Examples of such metadata are user name, organization, the date of creation of a data set, image, script or log, and the source, name/title, version and history (e.g., audit trail of what has been done to the data from its initial raw form) of a data set. Without conventions for nomenclature, this category is unsupported in context -- application software can only record and play back the text, not act on the stored information.

Evaluation criteria

As implementations of specific data models have matured and been used, experience in their use have led to a host of issues concerning actual applications. In this sense, an enumeration of criteria to evaluate extant implementations can be helpful. Much of the underlying issues were discussed in a workshop on Data Structures and Access Software for Scientific Visualization conducted at the ACM SIGGRAPH conference in Dallas, TX in 1990. From the report of that workshop these issues can be divided into two categories, access and implementation. In summary, the access issues relate to:

- How should data be brought into a visualization system?
- What does a system let a scientist do with the data?
- What does a system let an application programmer do with the data?
- How should data be described (i.e., metadata and data attributes as discussed in the previous section)?
- How does a system preserve the fidelity of the data?

Given implementation approaches as outlined in the Implementations and Techniques section of this paper, the issues that arise are:

- How are code AND data to be portable across multiple platforms?
- How is interoperability between structures to be maintained?
- At what (low-level) should interfaces be independent of the user?

The participants in the aforementioned workshop generally agreed that to address these issues, the following is required:

- Utilize well-defined layering
- Employ C for coding (although Java now shows potential as an alternative in the future) and provide bindings to other languages as required (e.g., FORTRAN)
- Maintain consistent terminology in interfaces and structures and use them to define a context for functionality

Scaling

A major concern for extant systems is in their practical application to a number of large data sets. These issues often relate to scaling to even modest data sets by today's standards independent of addressing raw bandwidth. The data effectiveness of a system can be measured by its ability to handle multiple data sets simultaneously of various sizes, types, structures, etc. without forcing artificial constraints that disrupt the fidelity of the original data. Systems that support different classes of data separately will have difficulty scaling to support disparate data properly at the same time. Systems that support different classes of data uniformly do not because they effectively decouple the management of and access to the data from the actual visualization software.

Another area of scaling is in performance related to aggregate data size and visual complexity (e.g., numbers of polygons, pixels, voxels, etc.). Given current and planned data rates for scientific investigations, whether computational or observational, workstation demonstrations with a few MB of data are hardly relevant. For real science problems, an effective system should be able to scale up to the storage capacity of a given platform (e.g., memory + swap) independent of the cpu performance. Therefore, the software should be able to generate images, for example, slowly on a small workstation, for what would seem to be big data sets for that platform while operations on small data sets would be more interactive. Higher performance should be achievable via increased aggregate cpu performance, which should include parallelism.

Therefore, one should consider the following:

- Data structure residency
  Are the data structures in an implementation on secondary storage (i.e., disk) and/or in primary storage (i.e., memory)?
- Transaction-like processing
  What types of access methods for data are supported? Is the ability to do DBMS-like operations on data provided, which can be important for data sets that are too expensive to even partially reproduce?
- Physical format and file structure
  How does an implementation compensate for typical limitations in conventional file systems (e.g., blocking) or operating systems (e.g., paging) for bulk data access?
- Distributed access to data within visualization systems

The implementations discussed in the Implementations and Techniques section of this paper address perhaps only a few of these points and are areas for which further research is required.

Conclusions

An effort to create data models and associated access software, especially to support scientific visualization involves the integration of various physical science disciplines and the computational sciences. It emphasizes the development of capabilities for a researcher to concentrate on doing science, freeing him or her from the mechanism of working with specialized data structures or formats. What is important is not the details of the technology, but what that technology can easily and inexpensively provide to promote science. Such activities will further the use of computer systems to support the management, analysis and display of any scientific data of interest under the control of the scientist doing research.

Additional reading

The following is a brief list of selected papers, documents, and other material related to scientific data structures, formats, access software and systems for data management and visualization. Apologies are given for any documentation, software or other material inadvertently omitted.

1. Brittain, D. L., J. Aller, M. Wilson and S.L. C. Wang. Design of an End-User Data Visualization System. Proceedings IEEE Visualization '90, pp. 323-328, October 1990.

2. Bancroft, G. V., F. J. Merritt, T. C. Plessel, P. G. Kelaita, R. K. McCabe and A. Globus. FAST: A Multi-Processer Environment for Visualization of Computational Fluid Dynamics. Proceedings IEEE Visualization '90, pp. 14-27, October 1990.

3. Brown, S. A. and D. Braddy. PDBLib User's Manual. Lawrence Livermore National Laboratory, Technical Report M270 Rev. 2, January 1993.

4. Brown, S. A., M. Folk, G. Goucher and R. Rew. Software for Portable Scientific Data Management. Computers in Physics, 7, n. 3, pp. 304-308, May/June 1993.

5. Butler, D. M. and M. H. Pendley. The Visualization Management System Approach to Visualization in Scientific Computing. Computers in Physics, 3, n.5, September/October 1989.

6. Butler, D. M. and M. H. Pendley. A Visualization Model Based on the Mathematics of Fiber Bundles. Computers in Physics, 3, n.5, September/October 1989.

7. Butler, D. M. and C. Hansen (ed.). Scientific Visualization Environments: A Report on a Workshop at Visualization '91. Computer Graphics, 26, n.3, pp. 213-216, February 1992.

8. Butler, D. M. and S. Bryson. Vector-Bundle Classes Form Powerful Tool for Scientific Visualization. Computers in Physics, 6, n. 6, November/December 1992.

9. Campbell, W., N. Short and L. Treinish. Adding Intelligence to Scientific Data Management, Computers in Physics, 3, n. 3, May/June 1989.

10. Campbell, W. J. and R. F. Cromp. Evolution of an Intelligent Information Fusion System.

Photogrammetric Engineering and Remote Sensing, 56, n. 6, June 1990.

11. Campbell, W. J., R. F. Cromp, G. Fekete, R. Wall and M. Goldberg. Panel on Techniques for Managing Very Large Scientific Data Bases. Proceedings IEEE Visualization '92, pp. 362-365, October 1992.

12. Cook, L. M. Mesh Types for Scientific Calculations. Lawrence Livermore National Laboratory, July 1990.

13. Dyer, D. S. A Dataflow Toolkit for Visualization. IEEE Computer Graphics and Applications, 10, n. 4, pp. 60-69, July 1990.

14. Faust, J. T. and D. S. Dyer. An Effective Data Format for Scientific Visualization. Proceedings of the SPIE/SPSE Symposium on Electronic Imaging, February 1990.

15. Fekete, G. Rendering and Managing Spherical Data with Sphere Quadtrees. Proceedings IEEE Visualization '90, pp. 176-186, October 1990.

16. French, J. C., A. K. Jones, J. L. Pfaltz. A Summary of the NSF Scientific Database Workshop. Quarterly Bulletin of IEEE Computer Society Technical Committee on Data Engineering, 13, n. 3, September 1990.

17. Gardels, K. Overview of Geographic Data Models and Geoprocessing. University of California at Berkeley Technical Report, December 1992.

18. Haber, R., B. Lucas and N. Collins. A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids. Proceedings IEEE Visualization '91 Conference, pp. 298-305, October 1991.

19. Hibbard, B. and D. Santek. The VIS-5D System for Easy Interactive Visualization. Proceedings IEEE Visualization '90, pp. 28-35, October 1990.

20. Hibbard, W., C. R. Dyer and B. Paul. Display of Scientific Data Structures for Algorithm Visualization. Proceedings IEEE Visualization '92, pp. 139-146, October 1992.

21. Hultquist, J. P. M. and E. L. Raible. SuperGlue: A Programming Environment for Scientific Visualization. Proceedings IEEE Visualization '92, pp. 243-249, October 1992.

22. International Business Machines Corporation. IBM Visualization Data Explorer User's Guide, Version 3 Release 1 Modification 4. IBM Document Number SC-38-0496-06, May 1997.

23. Kochevar, P., Z. Ahmed, J. Shade and C. Sharp. Bridging the Gap Between Visualization and Data Management: A Simple Visualization Management System. Proceedings IEEE Visualization '93, pp. 94-

101, October 1993.

24. Lang, U., R. Lang, R. Ruhle. Integration of Visualization and Scientific Calculation in a Software System. Proceedings IEEE Visualization '91 Conference, October 1991.

25. Li, Y. P., T. H. Handley, Jr., E. R. Dobinson. Data Hub: A Framework for Science Data Management. Submitted to 18th International Conference in Very Large Data Bases, August 1992.

26. Lucas, B., G. D. Abram, N. S. Collins, D. A. Epstein, D. L. Gresh, K. P. McAuliffe. An Architecture for a Scientific Visualization System. Proceedings IEEE Visualization '92, pp. 107-113, October 1992.

27. McCarthy, J. L. Scientific Information = Data + Metadata. Department of Statistics, Technical Report, Stanford University, March 1985.

28. NASA/OSSA Office of Standards and Technology. Definition of the Flexible Image Transport System. NASA/Goddard Space Flight Center, June 1993.

29. National Center for Supercomputing Applications. Hierarchical Data Format (HDF) Reference Manual V3.3. University of Illinois at UrbanaChampaign, February 1994.

30. National Center for Supercomputing Applications. HDF Vset, Version 2.0. University of Illinois at Urbana-Champaign, November 1990.

31. National Space Science Data Center. CDF User's Guide, Version 2.4. NASA/Goddard Space Flight Center, February 1994.

32. Pfau, L. M. The GridFile Tool Structure of System Files. Eidgen

33. Rasure, J. and C. Wallace. An Integrated Data Flow Visual Language and Software Development Environment. Journal of Visual Languages and Computing, 2, pp. 217246, 1991.

34. Rew. R. K. and G. P. Davis. NetCDF: An Interface for Scientific Data Access. IEEE Computer Graphics and Applications, 10, n.4, pp. 76-82, July 1990.

35. Salem, K. MR-CDF: Managing Multi-Resolution Scientific Data. Center of Excellence in Space Data and Information Systems Technical Report 9-28, NASA/Goddard Space Flight Center, March 1992.

36. Schroeder, W. J., W. E. Lorensen, G. D. Montanaro, C. R. Volpe. VISAGE: An Object-Oriented Scientific Visualization System. Proceedings IEEE Visualization '92, pp. 219-225, October 1992.

37. Silicon Graphics Computer Systems. IRIS Explorer. Technical Report BP-TR-1E01 (Rev. 7/91).

38. Smith, A. Q. and C. R. Clauer. A Versatile Source-Independent System for Digital Data Management. Eos Transactions American Geophysical Union, 67, pp. 188-189, 1986.

39. Stonebraker, M., J. Chen, N. Nathan, C. Paxson, A. Su and J. Wu. Tioga: A Database-Oriented Visualization Tool. Proceedings IEEE Visualization '93, pp. 8693, October 1993.

40. Treinish, L. A. and S. N. Ray. An Interactive Information System to Support Climate Research. Proceedings First International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, American Meteorology Society, pp. 72-79, January 1985.

41. Treinish, L. A. and M. L. Gough. A Software Package for the Data-Independent Storage of Multi-Dimensional Data. Eos Transactions American Geophysical Union, 68, pp. 633-635, July 14, 1987.

42. Treinish, L. A., J. Foley, W. Campbell, R. Haber and R. Gurwitz. Effective Software Systems for Scientific Visualization, Proceedings of SIGGRAPH '89 Panels, July 1989.

43. Treinish, L. A. An Interactive, Discipline-Independent Data Visualization System. Computers in Physics, 3, n. 4, July/August 1989.

44. Treinish, L. A. The Role of Data Management in Discipline-Independent Data Visualization. Proceedings of the SPIE/SPSE Symposium on Electronic Imaging, February 1990.

45. Treinish, L. A. (ed). Data Structures and Access Software for Scientific Visualization: A Report on a Workshop at SIGGRAPH '90. Computer Graphics, 25, n. 2, April 1991.

46. Treinish, L. A., D. M. Butler, H. Senay, G. G. Grinstein and S. T. Bryson. Panel on Grand Challenge Problems in Visualization Software. Proceedings IEEE Visualization '92, pp. 366-371, October 1992.

47. Unidata Program Center. NetCDF User's Guide, Version 2.3. February 1993.

48. Upson, C., T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz and A. van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. IEEE Computer Graphics and Applications, 9, n.4, July 1989, pp. 30-42.

49. Walatka, P. P. and P. G. Buning. PLOT3D User's Manual Version 3.6. NASA Technical Memorandum 101067, NASA/Ames Research Center, 1989.

50. Wells, D. C., E. W. Greisen and R. H. Harten. FITS: A Flexible Image Transport System. Astronomy and Astrophysics Supplement Series, 44, pp. 363-370, 1981.

51. Yamasaki, M. J. Distributed Library. RNR-90008, NASA/Ames Research Center, April 1990.

Selected representative scientific data structures, formats and access software

The accompanying table compares the characteristics of several fairly generic structures and software that are used to access and utilize scientific data in applications such as visualization. This list includes examples that provide a uniform data model, access mechanism or both, which have been used in visualization systems. Neither the characteristics nor the list is meant to be complete. It is meant to be a "living" document that serves as a point of reference on data structures, formats and access software for software developers as well as users and generators of data. Therefore, some of the information is likely to be already out of date. Hence, world-wide web and anonymous ftp addresses on the internet are provided for access to the latest material. There are innumerable omissions to keep the information succinct and useful, for which apologies are given. Some examples that are very domain-specific or no longer in widespread use or not readily available have been left out. Others are visualization or application environments/software, which may support many of the characteristics enumerated in the table. They have been omitted for either a lack of information from the developing organization or the support mechanism is not uniform. With regard to the latter, in other words, the software provides multiple paths for access to distinct data types rather than a single one. Many of the references cited in the previous section will address most of these examples.

Public domain contacts for information about data structures, formats, and access software:

Name	Contact	E-mail Address	Anonymous FTP	URL
Common Data Format (CDF)	Greg Goucher	goucher@nssdca.gsfc.nasa.gov	ncgl.gsfc.nasa.gov nssdca.gsfc.nasa.gov	http://nssdc.gsfc.nasa.gov/cdf/cdf_home.html
Flexible Image Transport System (FITS)	Don Wells	dwells@nrao.edu	fits.cv.nrao.edu	http://fits.cv.nrao.edu/
Hierarchical Data Format (HDF)	Mike Folk	mfolk@ncsa.uiuc.edu	ftp.ncsa.uiuc.edu	http://hdf.ncsa.uiuc.edu/
Network Common Data Form (netCDF)	Russ Rew	russ@unidata.ucar.edu	unidata.ucar.edu	http://www.unidata.ucar.edu/packages/netcdf/index.html

Portable Data Base (PDBlib)	Stewart Brown	sabrown@phoenix.ocf.llnl.gov	west.llnl.gov	http://www.llnl.gov/def_sci/pact/PACT_Docs/pdb/pdb5.doc.html

WorldWideWeb pages for information about visualization environments and associated data structures, formats and access software:

Name	**Contact**	**URL**
AVS	AVS	http://www.avs.com
Data Explorer	IBM	http://www.almaden.ibm.com/dx
FAST	NASA/Ames Research Center	http://science.nas.nasa.gov/Software/FAST/
IRIS Explorer	NAG	http://www.nag.co.uk/Welcome_IEC.html
Khoros	University of New Mexico	http://www.khoros.unm.edu/
SciAn	Florida State University	http://www.scri.fsu.edu/~lyons/scian/
VIS-5D	University of Wisconsin	http://www.ssec.wisc.edu/~billh/vis5d.html

lloydt@us.ibm.com

[DX Home Page | Contact DX]

Research home | IBM home | Order | Privacy | Legal | Contact IBM