Interactive Visualization of Earth and Space Science Computations

William L. Hibbard[1&2], Brian E. Paul[1], Andre' L. Battaiola[1&3],
David A. Santek[1], Marie-Francoise Voidrot-Martinez[4] and Charles R. Dyer[2]

[1]Space Science and Engineering Center
[2]Department of Computer Sciences
University of Wisconsin-Madison, 1225 W. Dayton St., Madison, WI 53706

[3]INPE/CPTEC, cp. 12201, Sao Jose dos Campos, Sao Paulo, Brazil

[4]Service Central D'exploitation de la Meteorologie, 31057 Toulouse Cedex, France

Keywords: visualization, interactive techniques, weather models, remote sensing

Abstract: We describe techniques that enable Earth and space scientists to interactively visualize and experiment with their computations. Numerical simulations of the Earth's atmosphere and oceans generate large and complex data sets, which we visualize in a highly interactive virtual Earth environment. We use data compression and distributed computing to maximize the size of simulations that can be explored, and a user interface tuned to the needs of environmental modelers. For the broader class of computations used by scientists we have developed more general techniques, integrating visualization with an environment for developing and executing algorithms. The key is providing a flexible data model that lets users define data types appropriate for their algorithms, and also providing a display model that lets users visualize those data types without placing a substantial burden of graphics knowledge on them.

1. Introduction

Computers have become essential tools for scientists simulating and observing nature. Simulations are formulated as mathematical models but are implemented as computer algorithms in order to simulate complex events. Observations are analyzed and understood in terms of mathematical models, but the volume of observations usually dictates that analyses be automated with computer algorithms.

In spite of their essential role, computers are also barriers to scientific understanding. Unlike hand calculations, automated computations are invisible and, because of the enormous numbers of individual operations in automated computations, the relation between an algorithm's input and output is often not intuitive. This problem was discussed in a 1987 report to the National Science Foundation,[1] and is illustrated by the behavior of meteorologists responsible for forecasting weather. Even in this age of computers, many meteorologists manually plot weather observations on maps, then draw isolines of temperature, pressure and other fields by hand (special pads of maps are printed for just this purpose). Similarly, radiologists use computers to collect medical data but are notoriously reluctant to apply image processing algorithms to those data. To these scientists with life and death responsibilities, computer algorithms are black boxes that increase rather than reduce risk.

The barrier between scientists and their computations can be bridged by techniques that make the internal workings of algorithms visible and that allow scientists to experiment with their computations. Here we describe how two interactive systems developed at the University of Wisconsin-Madison Space Science and Engineering Center (SSEC) are providing these capabilities to Earth and space scientists.

2. Visualizing numerical simulations of the Earth's environment

Numerical models of the Earth's atmosphere and oceans form one important class of scientific algorithms. The history files produced by these models are traces of their computations, and our VIS-5D (VISualization for 5-Dimensional data sets) system,[2] freely available by anonymous ftp and running on graphics workstations, is widely used by scientists for interactively visualizing these history files. This system takes it name from the fact that model history files are 5-D rectangles of data, organized as 2-D arrays of 3-D spatial grids. The 2-D arrays are indexed by time and by model field (e.g., temperature, pressure, salinity, three components of wind or current velocity, etc). Figure 1 shows the pipeline for rendering these data into 3-D animations under the user's interactive control. The data grids are transformed into graphical primitives that consist of 3-D vectors and polygons. On large workstations, we also use an efficient interactive volume rendering technique.[3] The rendering of graphical primitives creates a virtual Earth environment behind the workstation screen. Users can reach into this virtual environment with a mouse to move slices through the data grids, to place seed points for wind trajectories, and to rotate and zoom their view. In Figure 2 the window on the right contains the virtual Earth environment. The array of icons on the left lets users select combinations of fields and rendering techniques, and control animation, isolevels, trajectories, color maps, etc.

Modern workstations can respond to these controls within the time of an animation step (usually between 1/30 and 1/5 second), giving users the sense of interacting with a small virtual atmosphere or ocean. In order for users to explore the 3-D geometry of their fields, and to explore cause and effect relations between different fields, they should be able to rotate images and change the combinations of fields displayed without interrupting the smooth animation of model dynamics. Thus we do not synchronize animation with the computation of graphical primitives, rather storing primitives in intermediate tables indexed by time and by field.

The size of a model history file is the product of five numbers and can be quite large. For example, a data set spanning 100 latitudes by 100 longitudes by 20 vertical levels by 100 time steps by 10 model fields contains 200 million grid points. In order to maximize data set size we compress grid data and derived graphics by scaling them linearly to one or two byte integers. In order to preserve fidelity, different scaling factors are used for each horizontal slice of each 3-D grid. With compression we can store one grid point, plus derived graphics, in 2.5 bytes of virtual memory. For history files that are too large for workstations the system splits into a graphics client on a workstation connected via network to a data server on a supercomputer.[4]

Sometimes users need to see derived quantities, such as the vorticity or divergence of air flow, in order to understand the physics of a simulation. Users can write C and FORTRAN functions for deriving new diagnostic fields, and invoke them during a visualization session (they are dynamically linked with VIS-5D via sockets). In order to maximize data fidelity, floating point grid values in disk files, rather than compressed values, are used in these calculations.

2.1 Application experiences

William Gray of Colorado State University proposed a novel idea for generating energy by creating a permanent rainstorm over a hydro-electric generator. Figures 2 is a snapshot of a numerical experiment using this idea, performed by Gregory Tripoli and Peter Pokrandt of the University of Wisconsin-Madison using their UW-NMS weather model and visualized using VIS-5D. The white object is a balloon seven kilometers high in the shape of a squat chimney that floats in the air above a patch of tropical ocean. The purpose of the numerical experiment is to verify that, once air starts rising in the chimney, the motion will be self-sustaining and create a perpetual rainstorm. The vertical color slice shows the distribution of heat (and shows the flow of heat when model dynamics are animated), the yellow streamers show the corresponding flow of air up through the chimney, and the blue isosurface shows the precipitated cloud ice (a cloud water isosurface would obscure the view down the chimney, so it has been toggled off for this snapshot). The simulation takes many hours to run, even on the largest computers, so the virtual time of the visualization is not in lock step with the model's computations. Rather, model output accumulates in a history file, and users are free to move around in simulated time, searching for problems. Once found, users trace the root causes of problems by working back through time and by comparing different model fields.

Michael McCann and Matthew Koebbe of the Naval Postgraduate School retrieved VIS-5D by ftp and applied it to visualize the ocean simulation shown in Figure 3. This is a view from the north of a region of the Pacific ocean straddling the Equator, including ocean-bottom topography and a volume rendering of ocean current speed.

Simon Baas and Hans de Jong of Rijks Universiteit Leiden retrieved VIS-5D by ftp and modified the source code to allow VIS-5D to run solely under X Windows (i.e., without the GL library for 3-D graphics) and thus to run on a wide class of workstations. We enhanced their work and include it as an option with our standard ftp distribution. We also recently added support to VIS-5D for various non-Cartesian map projections and vertical coordinate systems.

VIS-5D is available by anonymous ftp from iris.ssec.wisc.edu (144.92.108.63) in the pub/vis5d directory. The README file contains complete instructions for retrieving and installing the software. The system includes source code and documentation. Although there is great interest in data format standards, they are only starting to be adopted by the scientific community. Thus we have found that the most important element for making our system usable has been a set of data import utilities. These include template programs (with versions in both C and FORTRAN) to help users convert their history files into a form that our software can read. Users modify these template programs to read their own history file formats.

3. Visualizing a broader class of computations

While VIS-5D is effective for visualizing simulations of the atmosphere and oceans, scientists design and use a much broader class of algorithms that require more general visualization techniques. We developed the VIS-AD (VISualization for Algorithm Development) system to meet this need.[5] Whereas VIS-5D runs as a post-process to simulations, VIS-AD serves as the execution environment for scientists' algorithms, supporting a greater variety of visual experiments with algorithms. VIS-5D assumes that data are organized as a five-dimensional rectangle, but in order to support a broad class of algorithms, VIS-AD enables scientists to define their own data organizations appropriate to their algorithms. Where the VIS-5D system's 3-D graphical space always represents 3-D physical space, VIS-AD allows scientists to define their own abstract graphical spaces. The VIS-AD system combines:

1. A **data model** that includes complex data types defined in terms of tuples and functional relations. The data model integrates several forms of metadata based on a conceptual model of computer data objects as finite approximations to mathematical objects.
2. A **computational model** based on a high level interpreted programming language that supports distributed computing and can link to user-written functions in C and FORTRAN.
3. A **display model** based on interactive, animated, 3-D voxel volumes. A novel technique enables scientists to control how their data are displayed without placing a substantial burden of graphics knowledge on them.
4. A **graphical user interface** that is highly interactive and that gives scientists an integrated view of data, computation and display.

The system functions like an interactive debugger with high-level data management and visualization. Where a debugger prints values of variables and arrays to help users track down low-level program bugs, VIS-AD generates visualizations of complex data objects to help scientists understand the high-level behavior of their algorithms. Coupled with a graphical interface for steering computations, these visualizations enable a wide variety of visual experiments with algorithms.

To scientists designing algorithms, the data model appears as a set of three rules for designing data types appropriate to their algorithms. Scientists may:

1. Define a scalar type for a primitive variable. Scalar types may be real variables like *time* or *ir* (an infrared radiance), they may be pairs or triples of real numbers like *lat_lon* (a pair of real numbers for the latitude and longitude of an Earth location), they may be integers like *count* (a frequency count used in a histogram), or they may be text strings like *satellite_id*.
2. Define a data type as a tuple of values of other types (this is like a *structure* in the C programming language).
3. Define an array type as a finite sampling of a functional relation from one type (the domain of the function - this must be a scalar type) to another type (the range of the function - this can be a complex type). An array data object is a finite set of objects of the range type indexed by values of the domain type.

Arrays and tuples can be combined in hierarchies to build complex data types. The left-hand column of Figure 4 shows how the rules may be applied to define two data types appropriate for a cloud discrimination algorithm. A data object of the *goes_sequence* type is a *time* sequence of satellite images, each partitioned into a set of rectangular *regions*. The image in each *region* is an array of pixels indexed by *lat_lon* values (the latitudes and longitudes of pixels' Earth locations). Each pixel is a tuple containing *vis* (visible) and *ir* (infrared) radiances, and *variance* and *texture* values computed by the algorithm from *ir* radiances. A data object of the *histogram_partition* type is an array of histograms computed by the algorithm, one in each image *region*, specifying frequency *counts* for the *ir* radiances of pixels in the *region*. Calculation of histograms is an important step in the cloud discrimination algorithm, and displays of these histograms are very useful for tracking down problems with the algorithm.

The center column of Figure 4 shows how users can control the displays of complex data objects by mapping scalar types to the components of the voxel-based display model diagrammed in the right-hand column of Figure 4. That is, users define mappings from scalar types to the *x*, *y* and *z* coordinates of voxels in the display, to the *colors* of voxels, to *animation* step number, etc. Because complex data types are ultimately defined in terms of scalar types, the system can derive depictions for complex data types from the mappings defined for their scalar components.

Figure 5 shows a data object of type *goes_sequence* displayed according to four different frames of reference. Its top right window shows the data object displayed as a colored terrain, as defined by the examples of scalar mappings in the center column of Figure 4. In the top left window both *ir* (red) and *vis* (blue-green) radiances are mapped to *color*. In the bottom right window *ir* is mapped to *selector* (only pixels whose *ir* radiances fall in the selected range are visible) and *time* is mapped to the vertical axis, producing a stack of four images. In the bottom left window *ir*, *vis* and *variance* are mapped to the three axes, *texture* is mapped to *color* and *lat_lon* is not mapped, producing a colored 3-D scatter diagram.

These displays are highly interactive. They can be rotated and zoomed using the mouse, they can be animated when scalars are mapped to *animation*, the mapping of scalars to *color* can be interactively adjusted using a color-map icon (an example is seen in Figure 7), and the contents of images can be changed using slider icons (examples are seen in Figure 8) when scalars are mapped to *selector*. The voxel-based display model fits naturally with volume rendering techniques,[6] and as graphics speeds improve the display model will be extended to include transparency and reflectivity values at each voxel. We will also add vector values at each voxel in order to provide a model for flow-rendering techniques.

Figure 6 illustrates the system's overall user interface via its application to a simple bubble sort algorithm. The window on the left is used to edit the text of the sort program. This program is written in an interpreted language similar to C - the syntax for user-defined data types is part of this language. Scientists' programs can call functions written in C or FORTRAN, and possibly running remotely across a network. Users can start and stop their programs, set breakpoints (by clicking on program lines), and execute single steps. They can also connect program values to graphical icons for interactively steering their computations. The dark horizontal bar across the program window indicates the current line of execution, and the short dark strings are the names of data objects that are selected for display. Data objects are selected by clicking on their names, and their depictions appear in the window on the right. The scalar mappings that define a display frame of reference are edited in the small text window at the top of the screen. Data may be displayed in several different frames of reference simultaneously (examples are seen in Figures 5 and 9).

The data being sorted in Figure 6 are an array of *temperatures* indexed by *time*. We have mapped *time* to the horizontal axis and *temperature* to the vertical axis, so the array is displayed as a graph (the set of white points) of *temperature* versus *time*. The bubble sort algorithm is organized as two nested loops. The index of the outer loop has type *time* and is displayed as a small green sphere on the lower horizontal axis - note that the white points to the right of the green sphere are sorted. The index of the inner loop also has type *time* and is displayed as a small red sphere - it marks the horizontal position of the current maximum value bubbling up through the array. The small blue sphere on the left hand vertical axis depicts an object of type *temperature* used as a temporary variable for swapping array values.

3.1 Integrating metadata into the data model

Mathematical models define infinite precision real numbers and functions with infinite domains, whereas computer data objects contain finite amounts of information and must therefore be approximations to the mathematical objects that they represent. Several forms of scientific metadata serve to specify how computer data objects approximate mathematical objects and these are integrated into our data model. For example, missing data codes (used for fallible sensor systems) may be viewed as approximations that carry no information. Any value or sub-object in a VIS-AD data object may be set to the *missing* value. Scientists often use arrays for finite samplings of continuous functions, as, for example, satellite image arrays are finite samplings of continuous radiance fields. Sampling metadata, such as those that assign Earth locations to pixels, and those that assign real radiances to coded (e.g., 8-bit) pixel values, quantify how arrays approximate functions and are integrated with VIS-AD array data objects.

The integration of metadata into our data model has practical consequences for the semantics of computation and display. For example, we define a data type *goes_image* as an array of *ir* radiances indexed by *lat_lon* values. Arrays of this data type are indexed by pairs of real numbers rather than by integers. If *goes_west* is a data object of type *goes_image* and *loc* is a data object of type *lat_lon* then the expression *goes_west*[*loc*] is evaluated by picking the sample of *goes_west* nearest to *loc*. If *loc* falls outside the region of the Earth covered by *goes_west* pixels then *goes_*west[*loc*] evaluates to the *missing* value. If *goes_east* is another data object of type *goes_image*, generated by a satellite with a different Earth perspective, then the expression *goes_west* - *goes_east* is evaluated by resampling *goes_east* to the samples of *goes_west* (i.e., by warping the *goes_east* image) before subtracting radiances. In Earth regions where the *goes_west* and *goes_east* images do not overlap, their difference is set to *missing* values. Thus metadata about map projections and missing data contribute to the semantics of computations.

Metadata similarly contribute to display semantics. If both *goes_east* and *goes_west* are selected for display, the system uses the sampling of their indices to co-register these two images in a common Earth frame of reference. The samplings of 2-D and 3-D array indices need not be Cartesian. For example, the sampling of *lat_lon* may define virtually any map projection. Thus data may be displayed in non-Cartesian coordinate systems.

3.2 Visualizing analyses of satellite observations

A pair of GOES satellites (located at East and West stations over the U.S.) generate one 1024 by 1024 image every four seconds. NASA's Earth Observing System, as planned, will generate about five 1024 by 1024 images per second. These data volumes are too large to be understood by direct visualization. Thus the proper role of visualization for satellite observations is helping scientists to develop algorithms for automating their analysis.

Robert Rabin of the National Severe Storms Laboratory, working at the University of Wisconsin-Madison, has developed algorithms for analyzing cumulus clouds in GOES satellite images.[7] These algorithms discriminate which pixels are part of cumulus clouds, and calculate a seasonal percentage of cumulus cloud cover as a function of time of day and location. The results of this computation are called a cloud census. We designed a *census_image* data type as an array of pixels indexed by *lat_lon*, where each pixel is a tuple containing a *cumulus_percent* and a *topography* (elevation of the Earth's surface above sea level). The cloud census is stored in an object of the *census_sequence* type, defined as an array of *census_image* data objects indexed by *time*. Figure 7 is a *census_sequence* data object displayed in a frame of reference defined by mapping *lat_lon* to the *XZ* plane, mapping *topography* to the *Y* axis, mapping *cumulus_percent* to *color*, and mapping *time* to *animation*. Note Lake Michigan in the upper right corner of the image. The color map icon in the upper left corner of Figure 7 shows that we have chosen purple for low cumulus percentages and blue for higher percentages. This display shows a clear correlation between cumulus percentage and topography, and when animated shows how cumulus clouds develop during the day. This is helping Rabin to understand how cumulus clouds form. Since ignorance of the mechanics of cloud formation is a major cause of uncertainty in efforts to predict the climatic consequences of the increase in greenhouse gases, such understanding may have important long term consequences.

3.3 Visualizing analyses of astrophysical observations

Because of the flexibility of its data and display models, VIS-AD is not limited to image processing applications. Figures 8 was generated from an algorithm for processing observations from an astrophysics mission. The Diffuse X-ray Spectrometer flew on the Space Shuttle in January 1993 and recorded several million events, each potentially an observation of an X-ray emanating from interstellar gas.[8] However, most of the recorded events are spurious, so Wilton Sanders and Richard Edgar of the University of Wisconsin-Madison needed to develop an algorithm for discriminating valid events. For this algorithm we defined the *xray_event* data type as a tuple containing scalars for an event's *time*, *wavelength*, *longitude*, *pulse_height*, *position_bin*, *goodness_of_fit*, *occulted_flag*, and many other fields. We also defined a data type *event_list* as an array of *xray_event* tuples indexed by *event_number*. Figure 8 shows a data object of the *event_list* type, displayed in a frame of reference defined by mapping *longitude*, *wavelength* and *time* to the three axes, mapping *pulse_height* to *color*, and mapping *position_bin* and *goodness_of_fit* to *selector*. In Figure 8, each X-ray event is displayed as a colored dot. Slider icons in the upper right corner are used to select ranges of values for *position_bin* and *goodness_of_fit*, and only those events whose field values fall in the selected ranges are displayed. This provides an easy way to experiment with event selection criteria.

In order to ferret out the mechanisms that produced spurious events, we defined many different frames of reference to see correlations among various sets of event fields. We also displayed the distribution of events as functions of various fields, as 1-D and 2-D histograms. The fact that we could change the display mappings of scalars as easily as we could rotate images was a key to successfully understanding the sources of spurious events.

3.4 Visualizing computations for education

Figure 9 was generated from a simple simulation of a 2-D cell of atmosphere. The dynamics of this cell of atmosphere are governed by a system of three differential equations developed by E. N. Lorenz[9] in order to study turbulence. Roland Stull chose to use this 2-D simulation in his course on Atmospheric Turbulence at the University of Wisconsin-Madison. The right window shows wind streamlines (isolines of the "stream function") and temperatures (warm air is red and cool air is blue) in the 2-D cell of atmosphere. The lower left window shows the solution to Lorenz's equations as a path through a three-dimensional phase space, revealing the two lobes of the familiar Lorenz attractor. The upper left window shows this same path in two phase space dimensions versus time, illustrating the apparently random (i.e., chaotic) temporal distribution of alternations between the two phase space lobes. The state of the 2-D atmosphere in the right window corresponds to a single blue point overlaid on the red phase space path in the lower left window. As the simulation algorithm runs, these displays of changing data objects animate the relation between the changing 2-D atmosphere and the blue point moving along the phase space path, showing that the two lobes of the Lorenz attractor in phase space correspond to clockwise and counterclockwise rotation in the 2-D cell of atmosphere.

4. Comparisons with other techniques

The data flow technique, represented by AVS, Iris Explorer and Data Explorer, gives users the flexibility to design their own rendering pipelines as networks of basic modules. We recognize the value of this approach, but designed VIS-5D with a fixed rendering pipeline diagrammed in Figure 1 that we felt could meet the needs of atmosphere and ocean modelers without asking them to design a module network. In fact, VIS-5D denies many choices to its users (e.g., shading model parameters, and colors and locations of light sources) in order to keep its user interface simple.

Because it interprets arrays as finite samplings of functional relations, the VIS-AD data model is similar to the data models of Data Explorer and SuperGlue that are based on fiber bundles. However, not all data models based on fiber bundles support complex hierarchies of tuples and functional relations, as the VIS-AD data model does. VIS-AD's scalar mappings define display functions that can be applied to any data type. This is similar to the polymorphic display functions defined in object oriented systems like SuperGlue and VISAGE. However, users of object oriented systems define display functions in a programming language, whereas users of VIS-AD define display functions by sets of scalar mappings.

Just as the data flow systems define a user interface for controlling data display based on the abstraction of the rendering pipeline, the VIS-AD system defines a user interface for controlling data display based on the abstraction of mappings from *scalars* to *display scalars*.

5. Conclusions

Interactive visualization techniques are making a difference in the work of scientists who have the means and make the effort to use them. We have exploited special assumptions about data organization to make it easy for scientists to apply VIS-5D to their data. The result is a system that is widely used by atmosphere and ocean modelers.

Scientists have needs that don't fit the special assumptions of VIS-5D, so we developed the VIS-AD system by generalizing some of the concepts of VIS-5D. Because of its flexibility, this system confronts its users with complex choices. However, we have organized these choices in a consistent framework of data, display and computational models. VIS-AD has demonstrated its utility to scientists working with its developers. When we complete its documentation and on line help functions, and make it available by ftp, we are confident that it will be useful to a wide community of scientists.

Acknowledgments

This work was supported by NASA grant NAG8-828, and by the National Science Foundation and the Defense Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives.

References

1. McCormick, B., T. DeFanti, and M. Brown, 1987; Visualization in Scientific Computing. Computer Graphics, 21(6).
2. Hibbard, W. and D. Santek, 1990: The VIS-5D system for easy interactive visualization. Visualization '90, San Francisco, IEEE, 28-35.
3. Hibbard, W., and D. Santek, 1989; Interactivity is the key. Chapel Hill Workshop on Volume Visualization, University of North Carolina, Chapel Hill, 39-43.
4. Hibbard, W., D. Santek, and G. Tripoli, 1991; Interactive atmospheric data access via high speed networks. Computer Networks and ISDN Systems, 22, 103-109.
5. Hibbard, W., C. Dyer, and B. Paul, 1992; Display of scientific data structures for algorithm visualization. Proceedings of Visualization '92, Boston, IEEE, 139-146.
6. Kaufman, A., D. Cohen, and R. Yagel, 1993; Volume Graphics. Computer 26(7), 51-64.
7. Rabin, R. M., S. Stadler, P. J. Wetzel, D. J. Stensrud, and M. Gregory, 1990; Observed effects of landscape variability on convective clouds. Bull. Amer. Meteor. Soc., 71, 272-280.
8. Sanders, W. T., R. J. Edgar, M. Juda, W. L. Kraushaar, D. McCammon, S. L. Snowden, J. Zhang, M. A. Skinner, K. Jahoda, R. Kelley, A. Smalle, C. Stahle, and A. Szymkowiak, 1993; Preliminary results from the Diffuse X-ray Spectrometer. EUV, X-ray, and Gamma-ray Instrumentation for Astronomy IV. SPIE, Vol. 2006, 221-232.
9. Lorenz, E. N., 1963; The mechanics of vacillation. J. Atmos. Sci., 20, 448-464 (138).