NASA-CR-200166

[illegible] NAG 8-828
IN-61-CR
[illegible]
8144 [illegible]

reprinted from
COMPUTER

# Interactive Visualization of Earth and Space Science Computations

William L. Hibbard, Brian E. Paul, David A. Santek, and
Charles R. Dyer, University of Wisconsin-Madison

André L. Battaiola, Instituto Nacional de Pesquisas Espaciais

Marie-Françoise Voidrot-Martinez,
Service Centrale d'Exploitation de la Météorologie

**Scientists often view computer algorithms as risk-filled black boxes. These visualization packages help scientists see the internal workings of their algorithms and thus understand their computations.**

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

In spite of their essential role, computers are also barriers to scientific understanding.[1] 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 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 that 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 two interactive systems developed at the University of Wisconsin-Madison Space Science and Engineering Center (SSEC) that provide these capabilities to Earth and space scientists.

## Visualizing Earth simulations

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 system[2] is widely used by scientists for interactively visualizing these history files. This system takes its name from the fact that model his-

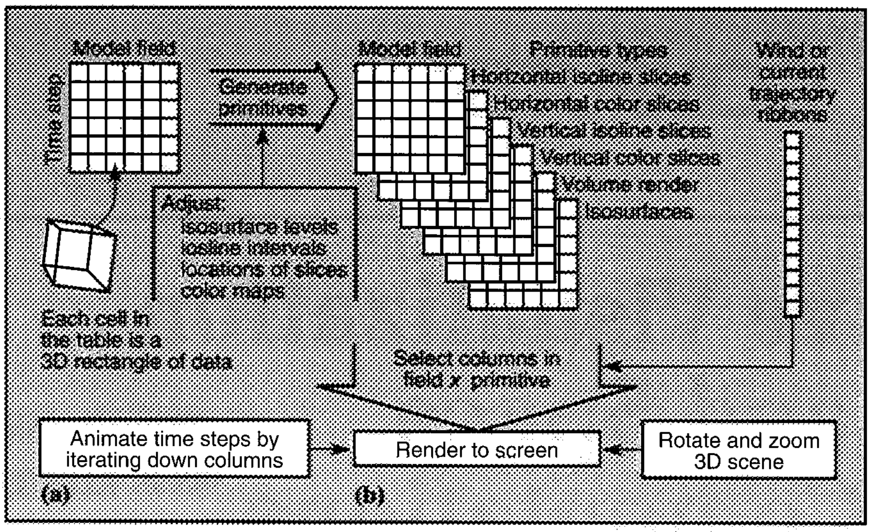


**Figure 1. Vis-5D transforms simulations of the Earth's atmosphere and oceans into an interactive graphical environment: (a) array of 3D grids indexed by time step and field; (b) array of graphics primitives indexed by time step, field, and primitive type.**

tory files are 5D rectangles of data, organized as 2D arrays of 3D spatial grids. The 2D arrays are indexed by time and by model field (for example, temperature, pressure, salinity, wind or current velocity, and so on).

Figure 1 shows the pipeline for rendering this data into 3D animations under the user's interactive control. The system transforms data grids into graphical primitives that consist of 3D 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, place seed points for wind trajectories, and 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 allows users to select combinations of fields and rendering techniques and to control animation, isolevels, trajectories, color maps, and so on.

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. To explore the 3D geometry of their fields, as well as cause-and-effect relationships between different fields, users 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; instead, we store primitives in intermediate tables indexed by time and 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. To maximize data set size, we compress grid data and derived graphics by scaling them linearly to one- or two-byte integers. To preserve fidelity, we use different scaling factors for each horizontal slice of each 3D 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 and a data server on a supercomputer connected via network.[4]

Sometimes users need to see derived quantities, such as the vorticity or divergence of air flow, 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). To maximize data fidelity, these calculations use floating-point grid values in disk files rather than compressed values.

To illustrate how Vis-5D works, Figure 2 shows a snapshot of a numerical experiment performed by Gregory Tripoli and Peter Pokrandt of the University of Wisconsin-Madison using their UW-NMS (Nonhydrostatic Modeling System) weather model and visualized using Vis-5D. They are modeling a novel idea proposed by William Gray of Colorado State University for generating energy by creating a permanent rainstorm over a hydroelectric generator. The white object is a balloon 7 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 (as well as 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-green 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 problems are found, users trace their root causes by working back through time and by comparing different model fields.

Michael McCann and Matthew Koebbe of the Naval Postgraduate School (NPS) applied Vis-5D to visualize the ocean simulation shown in Figure 3. This is a view from the north, looking at a region of the Pacific Ocean straddling the equator, including ocean bottom topography and a volume rendering of ocean current speed. Ocean models produce history files similar to atmosphere models, although the NPS model is remarkable for its high resolution and challenges the capacity of our visualization system.

## Visualizing a broader class of computations

While Vis-5D is effective for visualizing simulations of the atmosphere and oceans, scientists also design and use a much broader class of algorithms that requires 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 postprocess to simulations, Vis-AD serves as the execution environment for scientists' algorithms, supporting a greater variety of visual experiments with algorithms. Where Vis-5D assumes that data are organized as a five-dimensional rectangle and that 3D graphical space always represents 3D physical space, Vis-AD lets scientists define their own data organizations and abstract graphical spaces to support a broad class of algorithms. 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 3D voxel volumes. A novel technique lets scientists control how their data is displayed without placing a substantial burden of graphics knowledge on them.
(4) A *graphical user interface* that is highly interactive and gives scientists an integrated view of data, computation, and display.

The system functions like an interactive debugger with high-level data management and visualization. While a

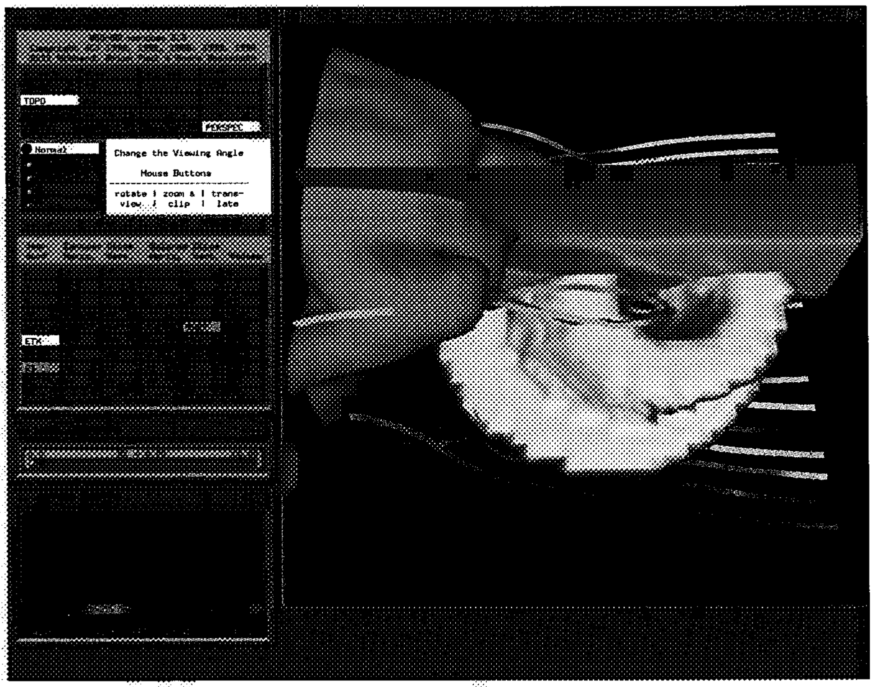


**Figure 2. Simulation of William Gray's novel idea to generate energy by creating a permanent rainstorm over a hydroelectric generator.**

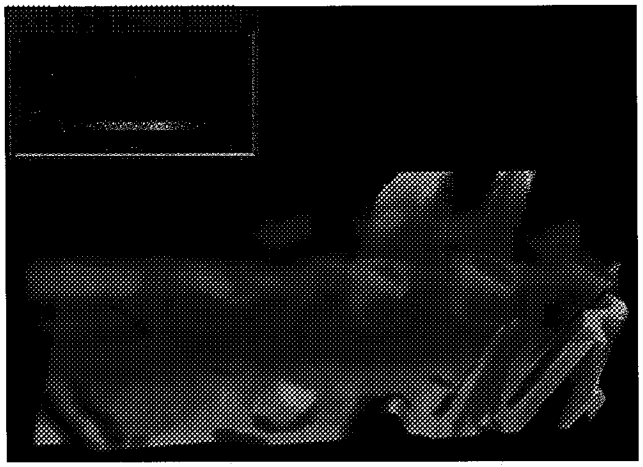


**Figure 3. Volume rendering of current speed from a simulation of the Pacific Ocean. The model's high resolution lets users see currents and eddies.**

## System availability

Vis-5D is available at no charge 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.

Simon Baas and Hans de Jong of Rijks Universiteit, Leiden, modified the Vis-5D source code so that on machines without special graphics hardware it can run under the X Window System (that is, without the GL library for 3D graphics) and, thus, on a wide class of workstations. We enhanced their work and include it as an option with our standard ftp distribution. We are also adding support to Vis-5D for various non-Cartesian map projections and vertical coordinate systems.

Although there is great interest in data format standards, the scientific community is only starting to adopt them. 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 can modify these template programs to read their own history file formats.

Vis-AD is also available by anonymous ftp from iris.ssec.wisc.edu. It is located in the pub/visad directory. Again, see the README file in that directory for complete information.

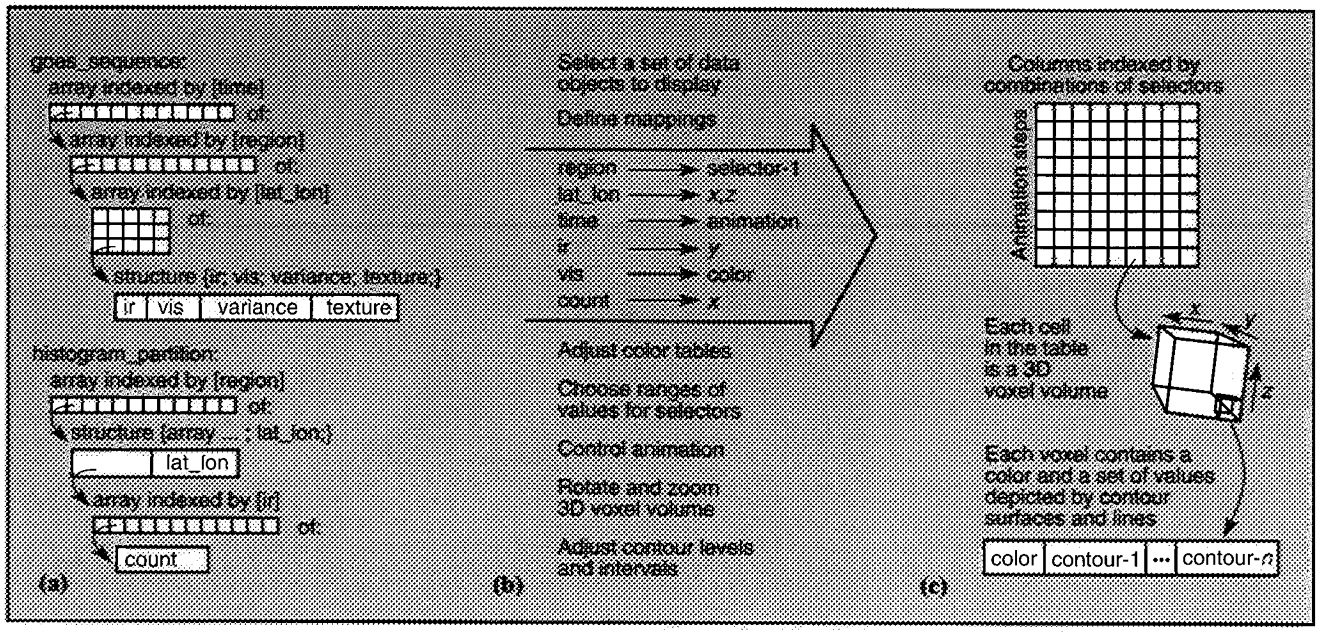

goes_sequence:
array indexed by {time}
of:
array indexed by {region}
of:
array indexed by {lat_lon}
of:
structure {ir; vis; variance; texture;}
ir
vis
variance
texture
histogram_partition:
array indexed by {region}
of:
structure {array _ ; lat_lon;}
lat_lon
array indexed by {ir}
of:
count
(a)
Select a set of data objects to display
Define mappings
region → selector-1
lat_lon → x,z
time → animation
ir → y
vis → color
count → x
Adjust color tables
Choose ranges of values for selectors
Control animation
Rotate and zoom 3D voxel volume
Adjust contour levels and intervals
(b)
Columns indexed by combinations of selectors
Animation steps
Each cell in the table is a 3D voxel volume
x
y
z
Each voxel contains a color and a set of values depicted by contour surfaces and lines
color
contour-1
...
contour-n
(c)

**Figure 4. The Vis-AD data and display models: (a) examples of data types that users can define in the data model; (b) user interface for controlling scalar mappings that control how data are depicted in the display model; (c) a diagram of the voxel-based display model.**

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.

**Designing data types for scientific algorithms.** To scientists designing algorithms, the data model appears as a set of three rules for designing data types appropriate to their algorithms. Scientists can

(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 can 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 the pixels' Earth locations). Each pixel is a tuple containing *is* (visible) and *ir* (infrared) radiances, as well as 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 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. 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, and so on. 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 and texture is mapped to color, producing a colored 3D scatter diagram (lat_lon is not mapped).

These displays are highly interactive. Users can rotate and zoom displays using the mouse, animate them, interactively adjust the mapping of scalars to color using a color-map icon, and change the subsets of data objects selected for display using slider icons. The voxel-based display model fits naturally with volume rendering techniques,[6] and as graphics speeds improve we will extend the display model to include transparency and reflectivity values at each voxel. We will also add vector values at each voxel 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 (the syntax for user-defined data types is part of this language). Scientists' programs can call functions written in C or Fortran, including those 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 selected for display. Users select data objects 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. The system can display data in several different frames of reference simultaneously (Figures 5 and 9 show multiple frames of reference).

The data object being sorted in Figure 6 is 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.

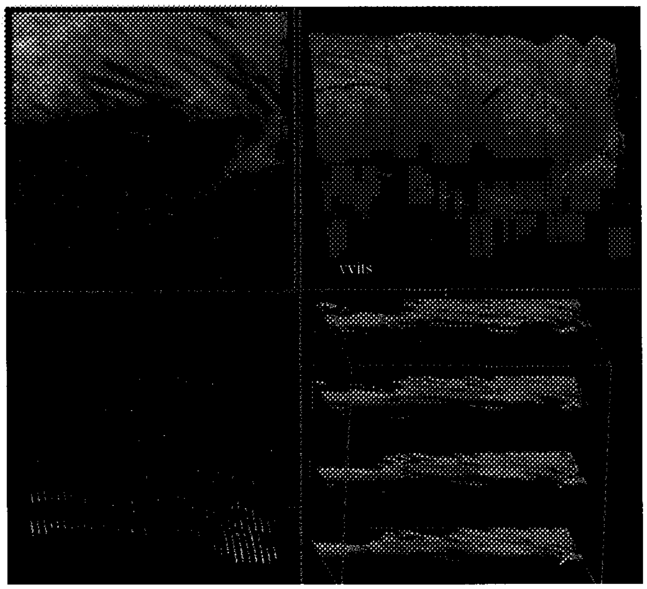
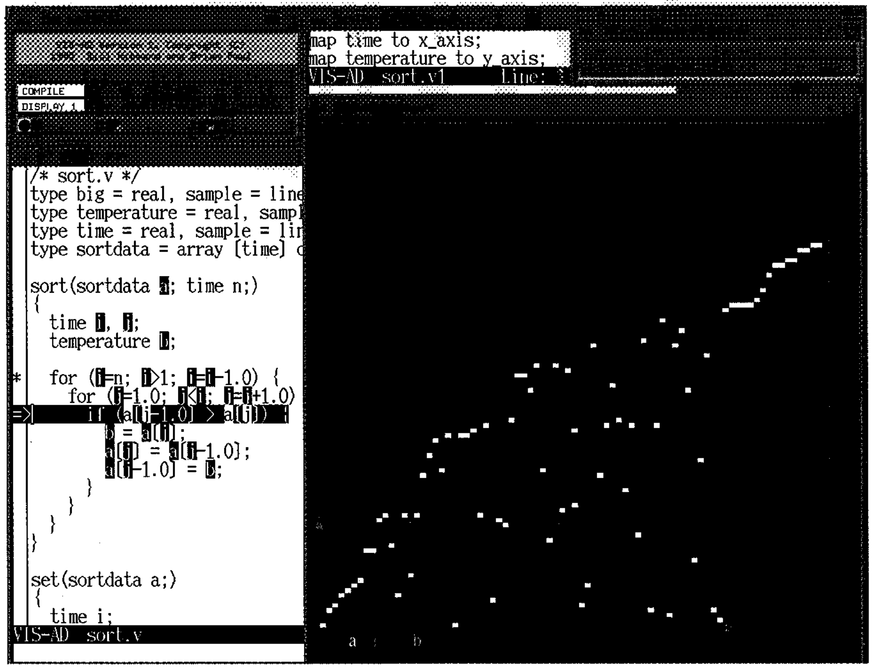


**Figure 5. A time sequence of satellite images displayed in four different frames of reference.**



```
map time to x_axis;
map temperature to y_axis;
```

```
/* sort.v */
type big = real, sample = line
type temperature = real, sampl
type time = real, sample = lin
type sortdata = array [time] c

sort(sortdata a; time n;)
 {
  time i, j;
  temperature b;

* for (i=n; i>1; i=i-1.0) {
    for (j=1.0; j<i; j=j+1.0)
=>      if (a[j-1.0] > a[j]) {
          b = a[j];
          a[j] = a[j-1.0];
          a[j-1.0] = b;
        }
    }
  }
}

set(sortdata a;)
 {
  time i;
```

**Figure 6. Visualizing the computations of a bubble-sort algorithm.**

**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 of the mathematical objects they represent. Several forms of scientific metadata serve to specify how computer data objects approximate mathematical objects, and we have integrated these into our data model. For example, missing data codes (used for fallible sensor systems) can be viewed as approximations that carry no information. Any value or subobject in a Vis-AD data object can 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 real radiances to coded (for example, 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

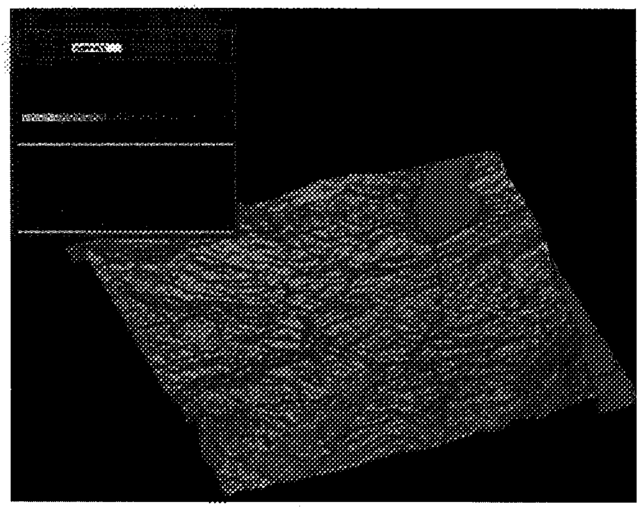


**Figure 7. Percentage of cumulus clouds derived from satellite data, mapped onto a Midwest topography.**

data type are indexed by pairs of real numbers rather than integers. If goes_west is a data object of type goes_image and *loc* is a data object of type lat_lon, the system evaluates the expression goes_west[loc] by picking the sample of goes_west nearest to loc. If loc falls outside the region of the Earth covered by goes_west pixels, 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 (that is, 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 contributes to the semantics of computations.

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

**Visualizing analyses of satellite observations.** A pair of Geostationary Operational Environmental Satellites (GOES) located at eastern and western stations over the US generate one $1,024 \times 1,024$ image every 4 seconds. NASA's Earth Observing System, as planned, will generate about five $1,024 \times 1,024$ 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 et al. of the National Severe Storms Laboratory, working at the University of Wisconsin-Madison, have developed algorithms for analyzing cumulus clouds in GOES images.[7] These algorithms identify 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 value (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 *x*-*z* 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 shows that we have chosen yellow for low cumulus percentages and blue for higher percentages. This display shows a clear correlation between cumulus percentage and topography, and when animated helps us to understand how cumulus clouds develop during the day. 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.

**Visualizing analyses of astrophysical observations.** Because of the flexibility of its data and display models, Vis-AD is not limited to image processing applications. Figure 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 identifying 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. The figure 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, by mapping pulse_height to color, and by mapping position_bin and goodness_of_fit to selector. 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, so that only those events whose field values fall in the selected ranges are displayed. This provides an easy way to experiment with event selection criteria.

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 in the form of 1D and 2D histograms. Our ability to change the display mappings of scalars as easily as we could rotate images was a key to successfully understanding the sources of spurious events.

**Visualizing computations for education.** Figure 9 was generated from a simple simulation of a 2D cell of atmosphere. The dynamics of this cell are governed by a system of three differential equations developed by E.N. Lorenz[9] to study turbulence. Roland Stull chose to use this 2D 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 2D cell of atmosphere. The lower left window shows the solution to Lorenz's equations as a path through a 3D 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 (that is, chaotic) temporal distribution of alternations between the two-phase-space lobes. The state of the 2D 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 2D 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 2D cell of atmosphere.

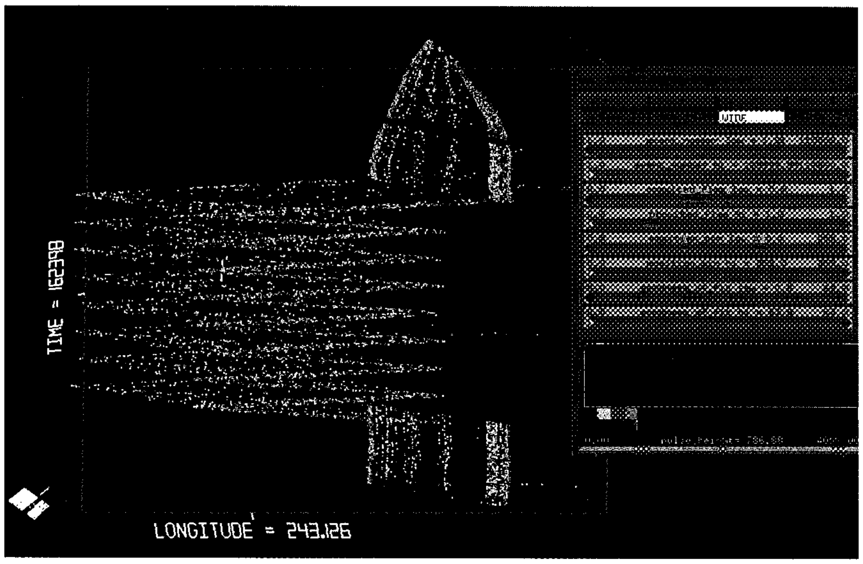


**Figure 8. X-ray events from a 1993 Diffuse X-ray Spectrometer flight on the space shuttle.**

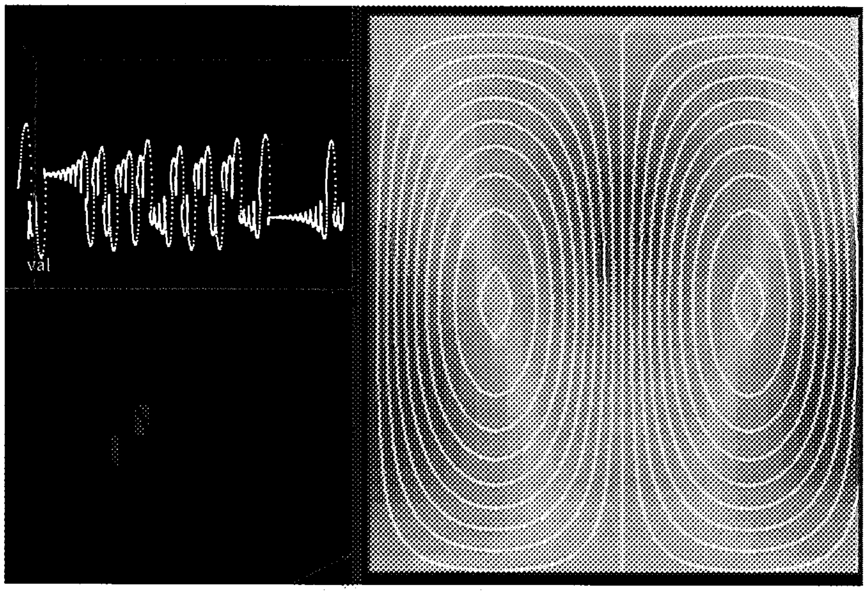


**Figure 9. Three views of the chaotic dynamics of the Lorenz equations.**

## Comparisons with other techniques

The dataflow technique — represented by AVS (Application Visualization System), Iris Explorer, and Data Explorer — gives users the flexibility to design their own rendering pipelines as networks of basic modules. Although we recognize the value of this approach, we designed Vis-5D with a fixed rendering pipeline (diagrammed in Figure 1), which 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 (for example, shading model parameters, and colors and locations of light sources) 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, which 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 dataflow 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.

Interactive visualization techniques are making a difference in the work of scientists who have the means and who 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 do not 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 online help functions, 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 Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives.

## References

1. B. McCormick, T. DeFanti, and M. Brown, "Visualization in Scientific Computing," *Computer Graphics*, Vol. 21, No. 6, Nov. 1987.

2. W. Hibbard and D. Santek, "The VIS-5D System for Easy Interactive Visualization," *Proc. Visualization 90*, IEEE CS Press, Los Alamitos, Calif., Order No. 2083, 1990, pp. 28-35.

3. W. Hibbard and D. Santek, "Interactivity is the Key," *Proc. Chapel Hill Workshop Volume Visualization*, Univ. of North Carolina, Chapel Hill, 1989, pp. 39-43.

4. W. Hibbard, D. Santek, and G. Tripoli, "Interactive Atmospheric Data Access Via High-Speed Networks," *Computer Networks and ISDN Systems*, Vol. 22, No. 2, Sept. 1991, pp. 103-109.

5. W. Hibbard, C. Dyer, and B. Paul, "Display of Scientific Data Structures for Algorithm Visualization," *Proc. Visualization 92*, IEEE CS Press, Los Alamitos, Calif., Order No. 3090-02, 1992, pp. 139-146.

6. A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics," *Computer*, Vol. 26, No. 7, July 1993, pp. 51-64.

7. R.M. Rabin et al., "Observed Effects of Landscape Variability on Convective Clouds," *Bull. Am. Meteorological Soc.*, Vol. 71, No. 3, Mar. 1990, pp. 272-280.

8. W.T. Sanders et al., "Preliminary Results from the Diffuse X-ray Spectrometer," *EUV, X-ray, and Gamma-ray Instrumentation for Astronomy IV*, Soc. Photooptical Instrumentation Engineers, Vol. 2,006, July 1993, pp. 221-232.

9. E.N. Lorenz, "The Mechanics of Vacillation," *J. Atmospheric Science*, Vol. 20, Sept. 1963, pp. 448-464.

**William L. Hibbard** is a researcher at the Space Science and Engineering Center of the University of Wisconsin-Madison. He is the principal investigator under the NASA grant that supported the development of the Vis-5D and Vis-AD systems and has been an investigator of the Blanca Gigabit Testbed. Hibbard has been a member of the program committees for the IEEE Visualization Conferences from 1990 to 1994.

He received a BA in mathematics in 1970 and an MS in computer science in 1974, both from the University of Wisconsin-Madison.

**Brian E. Paul** is a computer scientist at the Space Science and Engineering Center. His research interests include computer graphics and programming languages.

He received his BS in computer science at the University of Wisconsin-Oshkosh and is pursuing his master's degree at UW-Madison. He is a member of the ACM.

**David A. Santek** is a team leader in the scientific applications area at the Space Science and Engineering Center. His research interests include satellite data analysis, image analysis, and computer graphics.

He received a BS in atmospheric and oceanic science from the University of Michigan in 1975 and an MS in meteorology from the University of Wisconsin in 1978.

**Charles R. Dyer** is a professor in the Department of Computer Sciences at the University of Wisconsin-Madison. His research interests include computer vision, robotics, and visualization.

He received his BS degree from Stanford, his MS from the University of California at Los Angeles, and his PhD from the University of Maryland in 1979. He is on the Advisory Board of *IEEE Transactions on Pattern Analysis and Machine Intelligence* and is program cochair of the 1996 IEEE Conference on Computer Vision and Pattern Recognition.

**André Luis Battaiola** is a researcher at the Instituto Nacional de Pesquisas Espaciais (National Institute of Space Research) in Brazil. His research interests are computer graphics and scientific visualization.

He received his BS in physics and an MS and a PhD in electrical engineering from the University of Sao Paulo, Brazil, in 1982, 1987, and 1992, respectively.

**Marie-Françoise Voidrot-Martinez** is a computer scientist at Meteo-France. Her research interests include interactive computer graphics and user-interface design. She received a degree in meteorology from the French National Meteorological School in 1986 and a master's degree in computer science from the school of Centrale Paris in 1989.

Readers can contact William Hibbard at the Space Science Engineering Center, University of Wisconsin-Madison, 1225 W. Dayton St., Madison, WI 53706; e-mail whibbard@macc.wisc.edu.