Hurricane Mitch, Oct. 1998. Looking over Honduras northeast toward Cuba, this scientific visualization was created in 1999 using infrared and visible cloud data from the NOAA Geostationary Operational Environmental Satellites. Courtesy Gregory Shirah and Hal Pierce, NASA Goddard Space Fight Center.
Very powerful computers are a blessing to many fields of inquiry. They are also a curse; fast computations spew out massive amounts of data. Where megabyte data sets were once considered large, we now find data sets from individual simulations in the 300GB range. But understanding the data resulting from high-end computations is a significant endeavor. As more than one scientist has put it, it is just plain difficult to look at all the numbers. And as Richard W. Hamming, mathematician and pioneer computer scientist, pointed out, the purpose of computing is insight, not numbers.

Analyzing large amounts of data presents a number of technical challenges. Simply getting all the data for analysis stresses even high-end hardware; it can take an hour to load a 100GB data set into memory—assuming you have 100GB of memory to use. Loading the data little by little results in long times for a single pass through the data. These large data sets are interesting precisely because they contain new and often complex phe-
nomena. An overview of the data set can be crucial to understanding the phenomena they represent, as well as their context.

But once you have a sense of the phenomena to be found in the data, the actual information you are looking for can be extracted as a much smaller and more precisely focused subset of the data. This extraction can be performed through automatic processes that examine the entire data set.

Scientific visualization, or the use of computer graphics to represent data in ways that support understanding, has played an increasingly important role in the analysis of large data sets (see Figure 1 and Table 1). At NASA Ames Research Center’s Numerical Aerospace Simulation Division, we are developing visualization systems for understanding very large data sets. We have taken two basic visualization approaches in working toward this goal: browsing, or the ability to look at subsets of the data in a highly interactive way, and feature extraction, or the ability to automatically identify and exhibit specific phenomena of interest. Browsing requires a computer system that supports the rapid control, access, and display of user-specified parts of the data. These requirements drive aspects of the visualization system’s interface and architecture, as well as new ways of accessing data. Feature detection requires a clear understanding of the phenomena of interest and an algorithmic way of extracting the phenomena from that data.

Automatic feature detection techniques are by nature application specific, and computational fluid dynamics is the only area we are aware of in which feature detection has been deeply worked out. Our work at NASA Ames focuses primarily on computational fluid dynamics, a field characterized by very large time-varying simulations exhibiting extremely complex spatial phenomena, such as vorticity, turbulence, and stagnated flow (see the sidebar “Feature Extraction for Computational Fluid Dynamics”). While our goal is to service a specific domain, our methods have wide applicability, at least in the abstract. We hope readers from other domains working with large data sets find our experience helpful.

Browsing Complex 3D Data Sets

In light of the increasing complexity of 3D phenomena in simulations, 3D displays and interfaces provide a significant benefit for seeing and manipulating the forces that influence the behavior of virtual objects and the real objects they represent. This
benefit improves when the visualization system responds in near real time, typically frame rates exceeding 10Hz and latencies less than one tenth of a second. Such a quick turnaround allows for a “direct manipulation” interface, in which you place visualizations where you want them directly in 3D space. The result is a so-called virtual reality interface through which interaction with data visualizations can be like interacting with physical-world objects.

The payoff of such a responsive, intuitive interface is the ability to query data spatially, using visualization tools to explore a data set quickly. Such rapid exploration allows you to get a general sense of the data set and to concentrate on phenomena of interest. Such systems and the techniques used to build them are described in more detail in [1]; here, we concentrate on time and how to maintain responsiveness.

Visualization and analysis tools typically involve a simple three-step pipeline: the visualization is specified; the visualization graphical geometry is computed, accessing relevant data; and that geometry is rendered. Consider, for example, a streamline of a flow field through a point in space. A user interface specifies such points the streamline has to pass through, specifying the streamline. The streamline algorithm computes line elements of the streamline, accessing the vector field at each point along the streamline. When the streamline is completed by, say, reaching the end of the data set or reaching a prede-
fined length, the line elements are rendered in a 3D graphics system.

Near-real-time interaction requires this pipeline be run to completion within a tenth of a second. Longer times impede control in a direct manipulation interface. One way to maintain interactivity is to decouple the time scales of the interaction from the computation. Providing fast purely graphical tools that indicate how the visualization is specified provides a very responsive interface for the direct manipulation of slow visualizations. These considerations imply that the direct manipulation interface is run asynchronously from the visualization computation.

The visualizations themselves should be very responsive. Fast visualizations allow you to watch the visualization change as you move it about, exploring different regions of the data set (see Figure 2). But many visualizations, if unconstrained, take longer than a tenth of a second to compute. These considerations reflect the importance of constraining the visualization computation to a time budget, so it delivers a result within the time requirements of near-real-time interaction. Such constraints are called "time-critical computing" [2].

Time-critical computing applies ideas originally developed for computer graphics [7]. Its main purpose is assigning to a visualization computation a time budget and having the visualization compute the "best" answer it can within that budget. Two challenges characterize such a system: determining the allocation of the time budget to the various visualizations in an environment and the visualization algorithm deciding how best to meet the budget. The emphasis in time-critical graphics is finding a compromise between accuracy and performance. This compromise should not be confused with real-time computing, whereby specific tasks are performed to completion within a specified time. Time-critical computing balances many factors in an unpredictable environment, hopefully finding an appropriate compromise between such conflicting requirements as speed and accuracy.

Allocating the time budget to various visualizations is especially challenging. One wants to give more time to "more beneficial" visualizations in the interests of making them larger. In time-critical graphics, more beneficial is typically determined by such parameters as position on the screen and size of the object on the screen [7]. But estimating the benefit often requires computing the visualization first, because it may, for example, extend from off the screen into the center of the screen. However, precomputing a visualization takes time, defeating the purpose of the time-critical approach. This and other considerations make the a priori determination of the benefit of a visualization very difficult if not impossible in the abstract.

We assume that all visualizations are equally beneficial, augmented by information provided by the user. Thus, all visualizations in a particular environment are given an equal slice of the total allowed time (typically a tenth of a second, but determined by the user). If we then find the total time for all visualizations is too long or too short, we scale all visualization time budgets to yield the desired total time.

Once a visualization is given a time budget, the next challenge is to parameterize the visualization algorithm so it can do the best job in the available time. The easiest parameter to use is the "extent" of the computation to determine the size of the resulting visualization object. For streamlines, this extent is simply the length of the streamline, which keeps computing until it runs out of time. Use of an extent parameter implies a sense of locality; the visualization starts at a point in space and continues outward while assuming the starting point is of primary interest. Extent does not apply easily to conventional isosurfaces, which are specified by value rather than by spatial point; it does apply to local isosurfaces, which sample a scalar field value at a point in space and grow out from that point with an isosurface for that value.

Other highly visualization-dependent parameters can also be used to meet a time budget. For example, streamlines can be computed using a variety of algorithms of varying accuracy, suggesting that streamline algorithms choose, say, a less-accurate but faster algorithm to achieve a user-specified amount of time within the time budget [2]. The possibilities suggested by the plethora of visualization types are only beginning to be explored.

**Big Data for Scientific Visualization**

Interactively browsing a data set is less difficult when the data resides in physical memory. But many data sets are larger than the physical memory of a particular computer, and some data sets are much larger than any computer’s memory. We address this problem using techniques that reduce the amount of data that has to be handled by a visualization program. Techniques for providing interactive visualization can rely on sparse traversal, on compression, or on both. Sparse traversal represents the idea that a visualization algorithm analyzing scientific data may require only a subset of that data for each frame. Compression is the standard idea that a very large data set may have a smaller alternative representation that can be managed more easily at interactive rates.

**Sparse traversal.** To exploit sparse traversal, we have to find a visualization algorithm that requires
only a subset of the data. Not all visualization techniques have sparse traversal algorithms and are therefore a topic for further research. There are two classes of sparse traversal techniques: indices and data culling. Indices are created by preprocessing the data so the necessary parts can be identified at runtime. Data culling reduces the data requirements using other criteria, including whether or not the data is visible.

Isosurface extraction is a nice example of a visualization technique that historically required all data in the data set but for which sparse traversal algorithms were eventually developed. Isosurface extraction is the problem of finding the surface of some constant (iso)-value in a grid of scalar values. The first isosurface algorithm (developed by Lorensen and Cline in 1987 and called “marching cubes”) required online analysis of every cell in a grid to find those cells through which the isosurface passed. More recent algorithms require retrieval of only the cells of interest.

The newer isosurface algorithms, developed over the past seven years, provide offline processing to build an index into the data, so at run time, only those cells through which the surface passes have to be retrieved. Building indices is a common approach in database systems and can be applied just as profitably in visualization systems. We can generalize the isosurface index taxonomy of [5] to generally classify sparse traversal indices as: spatial, value, and seed. Spatial indices provide online lookup over the data set’s 2D, 3D, or 4D organizations. Value indices are content-addressable. Seed indices are the starting point for a visualization algorithm’s traversal of the data. These seed indices are developed primarily for isosurface extraction but offer an intriguing option for other visualization techniques.

Data-culling techniques induce sparse traversal. Applications can use view-dependent algorithms to reduce the amount of traversal needed to produce an image. For example, when walking through a city database, only the visible buildings have to be traversed. If hidden-surface culling is used, the traversal can be limited to the data not obscured by closer data. Another data-culling technique is to display data averages, instead of the data itself, when the data is too densely packed to be resolved. These output-resolution-dependent, level-of-detail, techniques work best with multiresolution techniques. While data-culling techniques are discussed extensively in the computer graphics literature, many visualization algorithms cannot use them because they traverse the underlying data rather than display the data directly. The underlying data has to be traversed because the data required for visualization may be completely unrelated to the user’s viewpoint and distance from the calculated 3D data.

Algorithms that perform sparse traversal can be improved with memory-hierarchy techniques similar to those used in operating and distributed systems. By keeping a cache of data in memory, frequently used blocks of data can reside in memory for quick access. Since visualization algorithms often reuse data, the cache can give performance approaching that seen when all the data is in memory.

An important aspect of optimizing performance is choosing the size and format of each block (see Figure 3). Each small square or rectangle in Figure 3 represents a block of data. Shaded blocks indicate the data read when the streamline is computed. The left-hand image represents the blocks and data read when the
original file format is used, where each block has a line of data (a plane of data in 3D). The right-hand image represents what is read when each block is a square (a cube in 3D). Less data would be read if the squares were smaller.

The advantages of sparse traversal and memory hierarchy techniques include the time required to compute a visualization of a simulation of the air flowing around an F/A-18 aircraft (see Table 2). Each row gives the time required for a run using a different amount of memory. The first column is the time required when sparse traversal is not used and all the data is read. The second column is the time required when sparse traversal is used with standard virtual memory and the original file format (in array order, using 16KB blocks). The last column is the time

Feature Extraction for Computational Fluid Dynamics

Scientists sometimes have a precise description of the phenomena they want to find in a data set. So there is little point in interactively exploring the entire data set when they can use that description to algorithmically find and indicate these phenomena while working on other things. This is the essential idea of feature extraction.

Examples of phenomena that are interesting and have mathematical descriptions in computational fluid dynamics include vortices, flow separation, and shocks. Using feature-extraction techniques to find them provides a precise, quantitative, and, above all, an automatic method for showing the phenomena of interest. Because automatic feature extraction can take a long time to access and process a large data set residing on disk, the computations are generally performed in a batch mode. But these computations have to be performed only once because the feature extraction algorithms search the entire data set and find every occurrence of the phenomena being sought. Feature extraction algorithms output 3D graphics primitives whose combined size is typically several orders of magnitude smaller than the original data set. This significant data reduction enables the results to be viewed interactively on modest graphics workstations. For these reasons, feature extraction has become a focus of visualization research.

The results of feature extraction are usually viewed in a 3D interactive visualization environment and can be used in conjunction with interactive visualization tools. These tools are used in parametric studies where tens or hundreds of simulations are run for the same aircraft with subtle changes to the aircraft’s structure or flight conditions. Moreover, as visualization and data analysis tools become more integrated in computational-fluid-dynamics flow solvers, information from them can be fed back into the flow solver programs to produce higher-fidelity solutions.

A number of feature extraction tools have been or are being developed to locate the following features most important to aerodynamicists.

Vortices. Vortices are readily identified by the rotating or swirling motion they create. They can cause such undesirable effects for aircraft in flight as airframe and control vibrations, even audible noise. Moreover, they can lead to structural fatigue and premature failure of the airframe under certain flight conditions. Despite being easy for the viewer of a visualization to recognize, there is actually no formal mathematical definition for describing a vortex, thus hampering development of related feature-extraction techniques.

Vortices can be detected automatically using a technique developed by Sujudi and the author Haimes at the Massachusetts Institute of Technology to locate the center of a swirling flow, commonly called a “vortex core,” using eigenvalue analysis of the velocity gradient tensor (see the Figure) [11].
required when application-controlled paging is used and the file is reformatted into 2KB blocks, each containing an $8 \times 8 \times 8$ cube of data. This data shows that sparse traversal improves performance, especially when memory is limited, and that controlling what is in each block improves performance even further. For more detail and discussion, see [3, 6].

**Compression.** While sparse traversal is primarily a property of visualization algorithms, “compressibility” is primarily a property of the data. Lossless and lossy compression can both be applied to scientific data. Lossless compression allows the original data set to be reconstructed; lossy compression does not. Much more research has focused on lossy compression than on lossless compression. While lossy compression of scientific data has been investigated primarily by the

The scheme works cell by cell, lending itself to distributed and parallel processing while requiring no user intervention to create an effective visualization with a minimum of geometry.

**Shocks.** Shock waves are generated by aircraft flying near or above the speed of sound. The environmental impact of shock waves is well known—as sonic booms. They also affect the aircraft itself, causing high loads on its skin, an abrupt increase in drag, and disturbance in the airflow. Shocks produce discontinuous changes in such flow-field quantities as velocity, pressure, temperature, and entropy.

Several feature-extraction techniques have been developed to extract shocks. The first, developed by Buning at NASA Ames Research Center, constructs isosurfaces of the normal Mach number, or the velocity divided by the speed of sound, resolved in the direction of the pressure gradient [12]. Shocks occur where the normal Mach number equals one.

**Flow separation and attachment.** Flow separation and attachment occur where a flow abruptly moves away from or returns to a solid body, including the surface of an aircraft. A separated airflow causes a significant increase in drag, increasing the aircraft’s stall speed, and is particularly dangerous at takeoff and landing, because it can cause the aircraft to stall and lose altitude rapidly.

Helman and Hesselink at Stanford University developed a visualization scheme for extracting separation surfaces based on the concepts of vector-field topology [9]. The topology of a vector field consists of critical points, or the points where velocity is zero, and tangent curves connecting these points. Streamlines are integrated out to the external flow, starting from points along the tangent curves and then triangulated to generate the separation surfaces.

Kenwright, one of the authors, developed a local technique for extracting separation and attachment lines based on eigenvalue analysis of the velocity gradient tensor [10]. This scheme detects two types of separation: open and closed. Closed separation lines originate and terminate at critical points, the type found by vector-field topology. Open separation lines do not have to start or end at critical points and are therefore not detected by vector field topology methods.

**Recirculation.** Recirculation is a phenomenon whereby fluid does not (quickly) leave a region. This flow structure is commonly called an eddy and is of great interest to designers of turbomachinery. In gas turbines, for example, stagnant hot fluid can significantly reduce the longevity of parts. Historically, researchers have generated streamlines or streaklines to identify regions of recirculation found when a streamline or particle is “trapped.” But the boundary of this region is difficult to compute for steady-state flows and impossible to compute for transient simulations.

A method that stays within the streamline/streakline definition of a separated region is called “residence-time” and computes the amount of time the fluid has been in (or in residence within) the domain. Because most of the fluid in a recirculation region stays inside that region for a considerable amount of time, the residence time of the fluid is much longer than that of the surrounding fluid. An isosurface can then be used to distinguish the region by solving a simple partial differential equation [8].

The computation times for these feature-detection algorithms can vary from seconds to hours, depending on the technique used and the size of the data set. However, the analysis can be done off-line in a batch computation. The feature-extraction techniques produce 3D geometry typically three to four orders of magnitude smaller than the original data set. Most important, these techniques give engineers the precise locations of all the flow features in their data sets without any human intervention.

7 Bryson 10 7/11/99 7:12 PM Page 89
computer science community, it has not generally been accepted by the scientific end user. Lossy compression is rejected most strongly by the medical community, and is viewed with skepticism by the engineering communities that must, say, design demonstrably safe airplanes based on simulation results. The most significant shortcoming of the work on lossy compression of scientific data has been the lack of error metrics demonstrating that no significant scientific or engineering information has been lost. The metrics acceptable for audio, video, and graphics fall short, particularly such metrics as image quality, signal-to-noise ratio, and root-mean-square error.

A more recent research thrust involves applying multiresolution techniques to scientific data. These techniques take as input the original data set and produce as output a transformed data set that is hierarchical—from coarse to finest representation of the data. “Wavelets” is the most popular multiresolution representation, capable of storing two successive levels of the hierarchy as averages and differences that can (in principle) be used to reconstruct the original data. Multiresolution representations potentially offer additional benefits over other compression techniques. One is that they allow “progressive refinement,” a technique whereby a lower-resolution representation can be browsed at interactive rates (because it is smaller); however, when the visualization algorithm requires finer detail, the higher-resolution data can be reconstructed incrementally.

A major drawback to multiresolution representations is that they are rarely lossless, because, in general, they achieve little overall compression. Instead, they typically compress the intermediate results of the wavelet transform, so accurate reconstruction is impossible. As a result, they suffer from the same lack of relevant error metrics as other compression approaches. Some recent research based on multiresolution analysis—feature-preserving transforms—is of interest in this regard [4]. The idea behind such transforms is that features of special interest to the scientist might be preserved by the compression, so a much smaller data set can be browsed without losing scientific insight. While this research direction shows promise, results are preliminary, and practical techniques of immediate applicability to the problem of very large data sets are not yet available.

**Conclusion**
Very large data sets are difficult for anyone to analyze and for computers to process. Two strategies help deal with this difficulty—interactive exploration and automatic feature detection—along with quick access to subsets of data from disk. A combination and automatic feature detection—along with analyze and for computers to process. Two strategies conclude very large data sets are not yet available.

References

Steve Bryson (bryson@nas.nasa.gov) is a research scientist at NASA Ames Research Center, Moffett Field, Calif.
David Kenwright (davidk@nas.nasa.gov) is a senior research scientist for MRJ Technology Solutions at NASA Ames Research Center, Moffett Field, Calif.
Michael Cox (mbc@nas.nasa.gov) is a senior research scientist for MRJ Technology Solutions at NASA Ames Research Center, Moffett Field, Calif.
David Ellsworth (ellswort@nas.nasa.gov) is a research scientist for MRJ Technology Solutions at NASA Ames Research Center, Moffett Field, Calif.
Robert Haines (haines@orville.mit.edu) is a principal research engineer in the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, Mass.

Most of the work described here was done in the Data Analysis Group of the NASA Ames Research Center’s Numerical Aerospace Simulation (NAS) Division. Work on feature detection was partially sponsored by NAS and by the Army Research Labs. Additional support came from the IBM SUR project and the IBM UUP project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 1999 ACM 0002-0782/99/0800 $5.00