Object-Oriented, Dataflow Visualization Systems—
A Paradigm Shift?

Chair: William Ribarsky, Graphics, Visualization, and Usability Center, Georgia Tech
Panelists: Bob Brown, Silicon Graphics Incorporated
Terry Myerson, International AVS Center
Richard Feldmann, National Institutes of Health
Stuart Smith, Institute for Visualization and Perception Research, University of Lowell
Lloyd Treinish, IBM T. J. Watson Research Center

Motivation and scope

Since the seminal paper on AVS by Upson et al. (CG&A, July 1989) just three years ago, there has been rapid development of several systems in this class, widespread use, and attempts to set up these systems as defacto standards for applications visualization. In some parts there is even talk that these methods have capabilities well beyond scientific visualization and open the door to construction of complete, integrated, interactive, and distributed approaches to the computing, manipulation, and analysis of all types of data. In this view, data from computer simulations and calculations of all sorts, from observations of any type, even from databases and program or network analysis, could be effectively manipulated and analyzed using these systems. Furthermore, the dataflow environment could be used to control the data production itself or to plan the best use of a distributed, heterogeneous computing system or observation and analysis network. All this from systems whose purveyors also claim them to be programmerless and usable by researchers themselves who would not need the services of graphics or visualization experts. Other workers in data visualization and analysis are not so sure about all the proclaimed benefits of the available object-oriented, dataflow approaches and see weaknesses and shortcomings that must be addressed. Still others think that different approaches are better or, at least, easier to use for certain classes of problems.

The past year has seen a significant expansion in the scope and number of dataflow visualization systems. Joining AVS and apE are the IRIS Explorer from SGI and the Visualization Data Explorer/6000 from IBM. In addition, apE, which at its peak had several hundred active users from government, industry, and academia, has been re-established under private ownership with plans for major extensions in the future. Similar systems, such as Khoros, a visual language environment applicable to many application domains, have also been developed and are gaining their own followings. Finally, the International AVS Center has just opened with a charge to expand the AVS user base and increase AVS functionality. Similar efforts can be expected from other vendors, and this activity promises to greatly extend the scope of these systems by, for one thing, collecting and making available user-supplied modules.

The purpose of this panel is to explore these issues by presenting contrasting views on the breadth and effectiveness of these systems and on the promise or capability of competing approaches. Among the questions that the panelists will address are:

- What is being done now with these systems?
- What are the prospects for the foreseeable future?
- Are there drawbacks in certain application areas? Basic limitations?
- What could be done to improve these systems? To make them even more usable, communicative, and extensible?
- Are there alternative analysis and visualization approaches that are better in some areas?

Panelist statements
Bob Brown

Large grain data flow systems for scientific visualization have proven to be very useful for qualitative investigations of moderately-sized data sets. They have not yet shown that the technique can be extended to very large data, or to areas outside the scientific domains. The challenge for LGDF visualization systems in the future will be to accommodate these new areas: very large data sets and visualization of other than traditional scientific data, such as financial or sociological profiles. Additionally, the paradigm may be highly useful for non-visualization tasks, since the composition technique used is generic and could prove useful in more general situations.
IRIS Explorer is an example of such a system, and is designed to be highly extensible. The primary idea behind Explorer is that it provides an infrastructure for program composition, without any policy concerning what programs can be composed. The primary development thrust of the system is to provide the mechanisms for program composition, allowing the user to determine the policy for application of the system to any task at hand.

**Terry Myerson**

Dataflow visualization systems are gaining popularity because of their efficiency in software sharing, module reuse, extensibility, and flexibility. The distributed capabilities of these systems have begun to address the issues of optimum resource usage and distributed database management. Few applications have fallen outside the breadth of these environments, and those only because of hardware constraints causing the need for specialized, optimized applications. Today, there is no more effective solution for the visualization industry than these systems.

Dataflow visualization systems will eventually provide the benchmarks for data navigation, animation, interactive steering of computations, application development and debugging, and remote access. Dataflow visualization environments are facilitating research in these areas at a much faster rate than would otherwise be possible with traditional environments.

What will distinguish one visualization system from the eventual defacto standard is going to be the effective distribution of the system's module components. The scalability, flexibility, and extensibility of these dataflow systems entirely depend upon these modules -- and their accessibility to the individual users is going to make the difference.

A challenge to each of these visualization environments is the graphical user interface. The ideal interface would be completely transparent to the user of the system -- allowing interactive, intuitive, and creative exploration of data without previous knowledge of the system. The dataflow paradigm has been established; but the graphical user interfaces for this paradigm have a significant period of evolution before them.

**Stuart Smith**

For those fortunate users whose needs nicely match the resources provided by a dataflow visualization system, these systems are truly marvelous tools. It is very gratifying just to link a few modules together, click on a button, and have an informative picture instantly appear on the screen. However, if even a single essential module is missing from the system's repertoire, using one of these systems can quickly turn into a nightmare. Unless the user has the services of a specialist developer who is thoroughly familiar with the system implementation and is able to code new modules from user specifications, the user must write the missing module. Thus, the user goes from having to know nothing about the system's implementation to having to know quite a lot about it, e.g., its internal data structures, its data type constraints, its data passing methods, its error handling, etc. The user's burden could be eased by well-designed module-writing tools, but current systems do not provide these.

A standard strategy for dealing with this situation is to try to anticipate every user's needs and provide appropriate modules. This "Swiss Army Knife" approach -- a blade for every conceivable purpose -- is apparently the strategy being followed by the AVS Center with its plans for 1000 modules. Under these conditions, the programmer's task is to find, and perhaps modify, the module that suits his or her needs. While this technique offers a way to avoid the continual reinventing of the wheel, it really is only as good as one's ability to find what one is looking for. Anyone who has attempted to learn a language like Smalltalk/V recognizes what a daunting task it can be to search the 200-page "Encyclopedia of Classes" for just the right object. Considerable experience is required to find and apply what one needs. Of course, issues of reusability and modifiability -- the hallmarks of OOP -- are meaningless if source code is not provided to the user.

Users of dataflow visualization systems fall into three categories: novices, intermediate users, and experts. Novices will load previously configured flow graphs to realize their desired visualizations, and they neither need nor want to know more about the system than is necessary to achieve them. Intermediate users will want to assemble modules in different ways to explore a variety of alternative visualizations. Existing dataflow systems have apparently been designed with this kind of user in mind. Experts will want not only to assemble existing modules into useful configurations but also to add new modules to the system; moreover, they will expect the new modules to fit together seamlessly with the existing ones. Only in the case of the fortunate intermediate user do current systems approach the "programmerless" ideal. Novices need someone to set up their graphs for them, and experts -- by definition -- program new system components.

Newer dataflow programming environments are including control flow "modules" such as IF, WHILE, and FOR in their visual programming systems. One look at such systems shows that the ability to program new modules and the ability to put them together in a variety of ways are necessary. We also need more complex event-handling. Often the user wants to take some input event
received within a display produced by a leaf module and redirect it to some module higher up in the graph. In addition, the user may want to handle events more complex than a simple mouse button press. For example, the user may want to select a complex region in order to pass it to some module that transforms the data or computes some image from the data. Such features indicate that dataflow systems are still evolving and that the existing popular systems are certainly not the last word.

Lloyd Treinish

As already cited there are several software systems available today that utilize the notion of "data flow". Since this panelist is from the group that has developed one such implementation (the IBM Visualization Data Explorer) it is critical to avoid overt commercialism in this discussion (i.e., us vs. them vs. the other guys). In that regard a focus on the effectiveness of such systems for addressing science problems is offered reflecting on some of the problems inherent in the study of large complex data sets that were considered as part of the development of that implementation.

Although not required as part of a data flow architecture, many implementations offer visual programming as the primary interface. This offers a powerful mechanism for programming at a high level of abstraction. However, it does not directly solve the primary problem of an end-user/scientist—that is, providing appropriate visualization mappings—the "have data want right picture(s) paradigm." This is the subject of another panel at this conference, "Grand Challenge Problems in Visualization Software."

It should be recognized that the greatest advantages of data flow tool kits and their typically associated visual prototyping. For the visualization-knowledgeable scientist or scientist who wishes to become knowledgeable in visualization, these can be directly applied in an end-user manner via program design or interactive manipulation of the programs in use. Despite the power and potential ease of use of such a style of interaction, it is often at odds with the procedural and programming (and often symbolic) nature of traditional data analysis. In addition, the sophisticated nature of the implementations of such data flow architectures may limit their extensibility for the average programming scientist. Such systems often can support relatively easy customization for specific research problems compared to more traditional, integrated software packages but always with more difficulty than independent code. It is essential for a scientist to experiment with new analysis algorithms and display techniques that will not be offered by the purveyor of the software. However, the method to implement such customization is likely to be one of standard programming in FORTRAN or C. This explains the popularity of software that offers a higher-level symbolic language for analysis over visual programming. Since data flow does not require visual programming, the support of both classes of interface by a data flow executive can be quite effective.

Independent of the underlying data flow execution model, a system may support the implementation of a traditional graphics user interface. Potentially this could include widgets for indirect manipulation and interactors within displays for direct manipulation. If the visual or other programming/customizing mechanism for the data flow system permits flexible specification and annotation of such an interface then it can potentially be used to build end-user applications. Otherwise, the application will appear to be too generic and hence, unfamiliar, or inflexible compared to the interface of an integrated software package, or simply a toy.

The major concern concerning the effectiveness of data flow systems either for end-user applications or in an interactive design mode for science problems relates to scaling. This notion of scaling can be decomposed into several areas. The first relates to data. Issues related to the role of data management in visualization will be addressed in the panel session on "Managing Large Scientific Databases" as well as the aforementioned panel on "Grand Challenge Problems in Visualization Software." However, the data effectiveness of a data flow system (or any system for that matter) can be measured by its ability to handle multiple data sets simultaneously of various sizes, types, structures, etc. without forcing artificial constraints that disrupt the fidelity of the original data. Systems that support different classes of data separately will have difficulty scaling to support disparate data properly at the same time. Systems that support different classes of data uniformly do not have this problem because they effectively decouple the management of and access to the data from the actual visualization software.

Another area of scaling is in performance related to aggregate data size and visual complexity (e.g., numbers of polygons, pixels, voxels, etc.). Given current and planned data rates for scientific investigations, whether computational or observational, workstation demonstrations with a few MB of data are hardly relevant. For real science problems, an effective data flow system should be able to scale up to the storage capacity of a given platform (e.g., memory + swap) independent of the cpu performance. Therefore, the software should be able to generate images, for example, slowly on a small
workstation, for what would seem to be big data sets for that platform while operations on small data sets would be more interactive. Higher performance should be achievable via increased aggregate CPU performance, which should include parallelism. Two architectural considerations are relevant. First, what is the execution model of the data flow system and the second, which is related, how is parallelism supported? A specific design choice may severely limit the ability of software implementations to effectively exploit newer computers, particularly with high bandwidth and/or multiple processors. Such high-performance machines are presumed to be required to support reasonable interaction on multiple, large data sets. The first consideration can relate to bandwidth. For example, a distributed execution model is well suited for supporting computational tasks (i.e., modules) running on a mixture of computer systems, independent of homogeneity. For large data sets, the ability to utilize multiple computers may not be practical because of insufficient bandwidth between processors. Depending on the implementation of the data flow executive, a distribution of independent processes on a single system may incur unacceptable overhead when scaled to large data sets even on a high-performance computer. A client-server execution model can avoid this problem because all of the data communications are in a single server process. The second consideration relates to computational performance. In principle, the use of parallelism is an effective way to improving performance cheaply. This is only true if the data flow system provides near linear speed up as one adds processors. If the software supports intermodule parallelism, which can be consistent with a distributed execution module, it may be very difficult to achieve efficient parallel execution even on a modest coarse-grain machine for more than a handful of processors. Intramodule parallelism is better suited to exploit such coarse-grain parallelism.

A third area of scaling relates to extensibility by the user. Customization (i.e., writing "modules") was cited earlier. However, for complex (i.e., real) applications, the ability to utilize an aggregate of hundreds or thousands of instances of modules in a data flow network is required, which obviously should include subroutines/hierarchy. Data flow networks with one or two screen "pages" of fairly simple modules generally are insufficient for providing user-level interaction with multiple, complex data sets. A purely distributed execution model where each instance of each module is a separate process would not scale well to network complexity required for realistic problems. A client-server execution model can scale easily for complex networks because all modules are available in the single server process.

Some of the aforementioned ideas have been exploited by this panelist via the IBM Visualization Data Explorer on a coarse-grain shared memory parallel computer (IBM POWER Visualization System) to do interactive browsing and analysis of large observational data sets and correlative visualization of multiple data sets in several earth and space science disciplines.

About the panelists

BOB BROWN has been at Silicon Graphics for two years, and was the system architect of the first release of IRIS Explorer. He holds a Ph.D. from Purdue University; his dissertation was in distributed program composition based on the large grain data flow model. Previous to his role at SGI, he was a research scientist at NASA. Currently, he is the engineering manager of the IRIS Explorer project.

TERRY JAY MYERSON is the technical staff member of the International AVS Center at the North Carolina Supercomputing Center. He has developed the U.S. EPA's third generation environmental modeling prototype using a distributed visualization environment, authored the International AVS Center's automated porting tool, and consults daily with the international research community.

WILLIAM RIBARSKY is Senior Research Scientist, Manager of the Scientific Visualization Lab, and Associate Director for Service of the Graphics, Visualization, and Usability Center, all at Georgia Tech. For several years he has carried out large-scale computer simulations (mostly molecular dynamics) of the atomistic properties of materials systems, liquid layers, and the interfaces between the two. This has led in recent years to research on the visualization of large, complex, dynamic and multivariate data, and on methods for their statistical and graphical analysis. Quite recently he has begun investigations of tightly integrated, large-scale simulations and data display using parallel computers and graphics workstations that will provide real-time visualization and interactive steering of computations. Dr. Ribarsky is a member of the American Physical Society and the IEEE Computer Society.

STUART SMITH is associate professor of Computer Science at the University of Massachusetts/Lowell. He holds music degrees from Rutgers and Brandeis, and taught music for 16 years at the University of Lowell. After receiving the Ed.D. in Computer-assisted Instruction (CAI) from the University of Massachusetts/Amherst in 1980, he joined the faculty of the Computer Science department at Lowell. An active hardware designer and consultant, Dr. Smith is co-holder of a U.S. patent for a robot sensor system. He has published articles on
scientific data visualization and sonification, computer-assisted instruction, and computer applications in sound and music. He is also the author of a computer science textbook. Dr. Smith's current research interests include scientific data visualization, interactive supercomputing, and the use of sound in user interfaces.

LLOYD A. TREINISH is a research staff member in the Scientific Visualization Systems Group in the Computer Sciences Department at IBM's Thomas J. Watson Research Center in Yorktown Heights, NY. He works on techniques, architectures and applications of data visualization for a wide variety of scientific disciplines. His research interests range from computer graphics, data storage structures, data representation methodologies, data base management, computer user interfaces, and data analysis algorithms to climatology, cartography, space plasma physics and planetary astronomy. Particularly, Mr. Treinish is interested in generic or discipline-independent techniques for the storage, manipulation, analysis and display of data. Earlier he did similar work in the development of advanced scientific data systems, including studying space and atmospheric phenomena, for over a decade at the National Space Science Data Center of NASA's Goddard Space Flight Center in Greenbelt, MD. A 1978 graduate of the Massachusetts Institute of Technology with an S.M. and an S.B. in physics, and an S.B. in earth and planetary sciences, Mr. Treinish has been at IBM since April 1990. He is a member of the IEEE Computer Society (IEEE-CS), the IEEE-CS Technical Committee on Computer Graphics, the Association for Computing Machinery (ACM), ACM SIGGRAPH, the National Computer Graphics Association, the Planetary Society, and the American Geophysical Union.