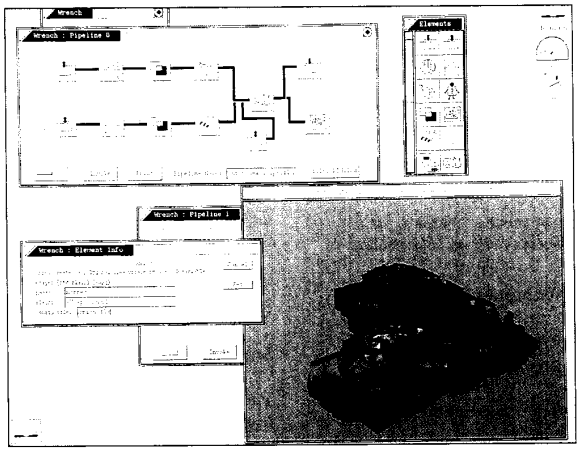
Visualization

A Dataflow Toolkit for Visualization

D. Scott Dyer
Ohio Supercomputer Center



The value of visualization in scientific computing is well established. The landmark report, *Visualization in Scientific Computing*,[1] published in late 1987, clearly presents the need for the visual analysis of information. We will not belabor this point with a discussion about the importance of visualization. Our focus, instead, will be on the process of building software to address this need. Rather than adapting an existing system to scientific visualization, we set out to create new software based on the needs of the user community. The toolkit described in this article is known as "apE." Originally, this was an acronym for "animation production Environment," but gradually apE came to be known as a software designed for more than just animation.

Visualization in scientific computing is not a new idea. Scientists have long known the value of pictures in the analysis of information. Our ability to generate scientific data, though, far outstrips our ability to turn that information into pictures. As the definition of visualization has widened to include everything from CAD to CFD, so too have the demands increased upon an embarrassingly weak graphics software base.

Advances in computer hardware dwarf advances in software technology. Large software systems can require years to complete and are obsolete long before they reach users. An increasingly complex array of problems could benefit from visual methods, but the software tools simply do not exist. We must build a software architecture that can grow with its users, that can be supported and maintained for many years, and

0272-17-16/90/0[illegible]0060$01.00 ©1990 IEEE

Due to a lack of contrast between text and background, this page did not reproduce well.

The History of the Ohio Supercomputer Center

In 1984, Ohio State University competed with institutions across the United States to host a National Supercomputer Center. While its proposal was highly ranked, Ohio State lost its bid for National Science Foundation funding to obtain a center. However, the highly motivated group of computational chemists that spearheaded Ohio State's efforts then received help from the state. In 1987, the legislature appropriated funds for the Ohio Board of Regents' supercomputer initiative to create a center serving academic and industrial users in the state of Ohio. In June 1987, a Cray X-MP was installed at the Ohio Supercomputer Center, followed by a Cray Y-MP in July 1989.

One of the early supporters of the Ohio Supercomputer Center was Professor Charles Csuri, a pioneer in the field of computer graphics and Director of the Advanced Computing Center for the Arts and Design at Ohio State. He foresaw the rise of scientific visualization in the early eighties and built a significant graphics research component into the base of the then fledgling Ohio Supercomputer Center. Today, 10 computer graphics professionals comprise the Ohio Supercomputer Graphics Project.

OSGP members include, from left to right (top row): H. Stephen Anderson, John C. Donkin, and Jeffrey T. Faust (software), Jill L. Kempf and Robert E. Marshall (applications), and (second row) John Andrew Berton Jr. and Peter G. Carswell (animation). In addition, Michelle Messenger provides project coordination and user support and Barbara Helfer-Dean operates the Ohio Visualization Laboratory. Stephen N. Spencer and Jeffrey Light (front row) from the Advanced Computing Center for the Arts and Design also provide software in the construction of apE. Joan Staveley, Chitra Sriram (not pictured), and Mark LeScoezec (not pictured) provide documentation support and help to operate the Ohio Visualization Laboratory.

that can keep pace with changes in hardware—in short, we must develop a new software methodology to meet the challenge of visualization.

Previous work

The rise of visualization as a computer-graphics buzzword has generated an enormous amount of work, some of it very good, much of it simply adaptations of old research in other areas, retitled and resubmitted. Much current work is centered around the notion of dataflow as an ideal abstraction for the visualization process. The Application Visualization System[2] is in many ways remarkably similar to the software described in this article. Both AVS and apE were constructed at about the same time, though with little communication between the developers. While the same dataflow decomposition is used, along with a visual programming paradigm, significant differences in construction, data elements, and portability are apparent to users familiar with both AVS and apE. Researchers at the State University of New York at Stony Brook have constructed a system, known as VERITAS, aimed more at the user interface for scientific visualization.[3] Zabusky and Bitz have built a system known as DAVID that encompasses the interactive and multimethod visualization techniques common to dataflow systems.[4] Commercial systems such as PV-WAVE[5] present alternative, albeit more traditional, approaches to the visualization problem.

Designing a beast

In late 1987, the Ohio Supercomputer Graphics Project set out to design an effective software system for visualization. Rather than dictate to the scientific community a particular methodology, we spent extensive time with potential users to understand the real needs of scientific research. We listened to users of all kinds of current graphics software and hardware. We discovered the realities of fixed budgets that permit only modest hardware acquisition and the effects of slow network connections on high-speed computing. In short, we tried to face the real world, and to design and build a product that would outlast current hardware platforms while providing a high degree of flexibility to today's users.

The construction of a large graphics system is as much a matter of avoiding the mistakes of the past as breaking new ground in software engineering. McCormick et al.[1] jokingly refer to "doorstop manuals," those large, bulky "reference" books that are common among many of the most popular graphics software systems today. We wished to avoid constructing a large, monolithic library; we wished to avoid implementing lots of separate, difficult programs and data

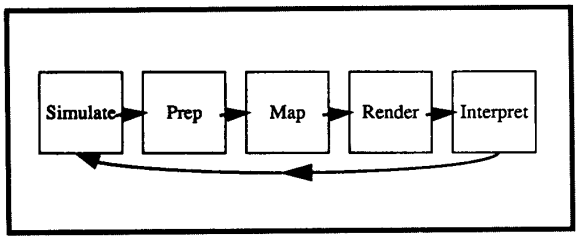


Figure 1. The visualization pipeline.

formats; we wished to avoid building a system whose life span was less than our interest in the project.

Efforts in the mid-eighties at the Computer Graphics Research Group (now known as the Advanced Computing Center for the Arts and Design) at Ohio State University led us to select a dataflow model for the apE system. Dataflow maps very well to the general steps followed in visualizing scientific data. Most researchers follow a five-step process, beginning with a computational or experimental simulation, and concluding with interpretation.[6] Intermediate steps include preparation, mapping, and rendering (the prep stage is occasionally omitted or merged with the map stage). Ideally, the results of interpretation can be fed back into the original experiment or simulation. This kind of feedback is known as steering, and has been used with great success in limited applications.[7] New software technology is needed to investigate the steering issue completely.

The mapping stage is the key to the visualization pipeline, shown in Figure 1. Often ignored, it is clearly the hardest step. Data preparation includes normalization or other mathematical steps already well known to most researchers, and rendering is in the domain of the graphics community and has been nearly beaten to death in every variety. The mapping stage is the natural juncture of scientific and graphical data. Vendors often proclaim certain workstations or software products to be ideal for visualization, because they can produce X polygons per second or Y vectors per minute, but they are really only solving one step in the pipeline, and not providing a visualization solution. Visualization is much more than high-quality rendering, radiosity, or real-time texture mapping. For the needs of most scientists, the computer-graphics community has solved the rendering stage of the pipeline. What the graphics community has not provided to date is an effective method for converting generic scientific data into graphical forms. The dataflow model is an attractive abstraction because it naturally highlights the mapping stage and can provide a powerful mechanism for interactive exploration of a variety of mapping methods (see Figure 2).

The dataflow abstraction is ideal for remote execution and parallel operation. Network computing environments are commonplace, and distributed computation is a requirement for maximum resource use. Dataflow systems can naturally distribute each execution element on a separate machine or processor. Because our notion of dataflow is data-driven and not demand-driven, we also get the benefit of parallel execution for time-dependent or multiframe data sets. Each element operates not under the control of some central authority but only as input data, frame boundaries, and other local conditions dictate. Successive elements in a visualization pipeline can be operating on separate groups of data, all in concert, without any additional user interaction. This notion of distributed computing reflects the situation researchers often face—finding themselves far from their supercomputers but possibly near some local computing power.

Once we were firmly committed to the dataflow concept, we examined the requirements for a data language. Incompatible binary formats are common in a heterogeneous network environment. While transmission of data as text files would mitigate this problem, the operational overhead for such transmissions was out of the question in an interactive system. Thus a dataflow language was born, designed not only to represent common data elements from the scientific domain (such as grids and variables) but also such common forms as objects, images, and geometries.

With the system mapped out, we began to plan its implementation. We recognized early on that traditional programming methods for design, communication, and maintenance would be insufficient for this effort. It was also clear that a project of this size could not be fully planned and documented prior to beginning the work, especially with a limited staff and budget. The advantages of working within the academic world are enormous, but it meant we would have to design and implement the system as well as test, document, and support it.

Building any beast

Our approach in determining the needs and goals of the software system prior to implementation may sound obvious, but it is often abandoned in favor of rapid, market-driven development. We were guided by four simple principles.

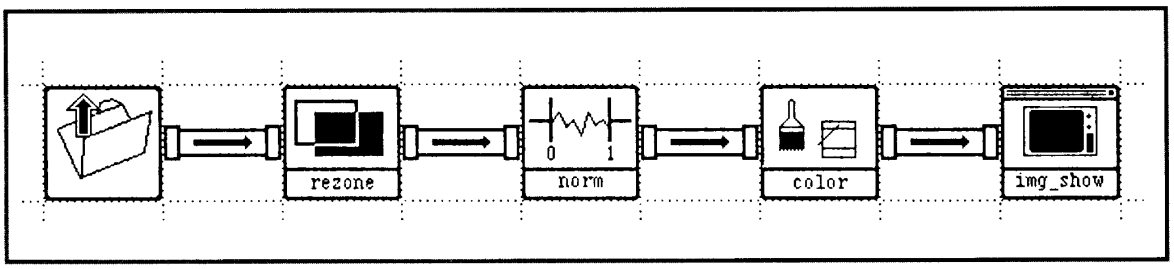


Figure 2. An example visualization pipeline from apE 1.1. Data moves from the simulation, left to right, and is rezoned, normalized, colored, and displayed.

1. *Build the software around the users.* One fundamental mistake that is often made in planning and executing a large software system is an initial failure to gauge the needs of the eventual users adequately. For example, graphics software is often (quite naturally) designed to display graphical data, such as patches, polygons, and vectors. In the case of scientific visualization, however, such entities are both potentially unknown to users and fundamentally irrelevant. Visualization software must be designed around the users' data, not graphical data. This concept extends naturally into all facets of software development. Developers must keep in touch with their user community.
2. *Base the software on market reality, not hype.* It is tempting to design software around ideal platforms or for systems that do not exist. Too often, we develop systems that require exotic configurations or work on only a small number of existing systems. This applies not only to hardware but software as well—for example, everyone claims to have graphics library A or B and high-level window system C or D—but few actually ship it as a part of a normally configured system. It is important to take the time to see things as potential users see them, not as software developers with nearly infinite budgets.
3. *Let the users control the software.* Longevity is achieved only by allowing the user community to control the software. This does not necessarily mean that the developers must relinquish complete control, only that the users must have a stake in the software development. This translates to an active user community, availability of source, and a strong willingness to tailor the software to the users.
4. *Distribute it widely and cheaply.* This point cannot be emphasized enough. Software is a vital productivity resource, and needs to be widely available. Restrictive licensing or an attempt to profit from the development strongly limits the downstream success of the project. The long-term benefits of a loose distribution policy are tremendous. Even for profit-making companies, it is better to sell several thousand copies of the software at a low price than several dozen at a high price.

Kawasaki[8] lists four key elements in any successful product: depth, indulgence, completeness, and elegance. Deep products satisfy all levels of users, from the novice to the expert. Indulgent products are often overkill for most of the tasks they are used for. Complete products provide all levels of user service, infrastructure, and growth path. Elegant products are both creative and inspiring—they seem to achieve their functions almost transparently—and are easy to learn and use. ApE was designed to be deep, indulgent, complete, and elegant. By providing complete documentation (and tutorials), we address the novice, but the presence of source code guarantees the interest of the expert. ApE contains a complete interface design system, data language, and source-code management system, as well as significant tools for graphics at all levels. ApE provides user services through a user group, electronic mail, documentation, and a desire to redistribute the extensions that come from the users. ApE is based on a model that allows the exotic application to benefit fully from visualization—transparently distributing execution and permitting the best use of local resources.

A set of development objectives such as these are valuable for establishing the "feel" of a project. Those developing the system must "buy" into it at an early stage, and believe in it as gospel as the project rises and falls through its history. ApE has been under development for more than three years, and has perhaps 10 or more person-years invested in it. Over a million

Convex (C-1)	Silicon Graphics Personal Iris
Cray X-MP, Y-MP	Silicon Graphics 4D Series
Decstation 3100	Stardent GS1000/2000
HP 300 & 800 series	Sun 386i
Macintosh II (A/UX)	Sun 3
Next	Sun 4

Figure 3. Machines currently supported in apE development.

lines of code have been written; many have been discarded. To date, not one of the original developers has left the project. One measure of a system's value is purely technical—does it solve a problem or achieve the correct result?—but another equally important measurement criterion looks at longevity and use. The chief goal of the Ohio Supercomputer Graphics Project is to create a usable, stable, long-lived environment for scientific visualization.

Building a large beast

As all programmers know, there is a great difference between writing a small piece of personal software and constructing a large software environment. The challenge of machine and device independence and portability made our task more difficult, as did the demands of making a software that is to be distributed not as a closed system but as source code to be modified, improved, and extended as required.

Using existing software and hardware technology, we tried to build a graphics environment that is as portable as possible. Clearly, measures of portability have changed since 1987, when we began this project. However, many of the design decisions we faced then are still faced today by large-scale developers. These decisions can be summarized as three primary turning points: the selection of an operating system, the selection of a graphics library, and the selection of a user interface.

On the Unix platform

We chose to build our system on the Unix platform. The mid-eighties saw an explosive growth in a new breed of personal computer known as the "workstation." Performance, power, and software resources that were once only part of large mainframe systems rapidly became available on the desktop, and the Unix operating system quickly became the de facto standard operating system. Manufacturers who did not, could not, or chose not to respond to this trend saw their sales diminish.

Implementation under Unix has its share of difficulties, however, because Unix is not a standard. An apE library is devoted to hiding the various differences and peculiarities unique to each version of Unix we encountered. The software is implemented entirely in C, with some Fortran extensions to support scientific users. While Objective-C[9] and C++[10] both offer advances in software engineering and would aid in this development, we chose not to use them because neither has yet achieved a sufficient following to be declared a de facto standard and because neither is typically found on a generic workstation.

Our decision to implement apE on multiple Unix platforms required a carefully planned system for source, object, and executable code storage. A series of scripts were designed to provide top level management tools for the software environment known as forge. The forge technology allowed us to handle machine dependencies, multiple source files, and recompilation dependencies all from a single set of declarations. A single command is capable of recompiling all source code on all supported systems, without any user intervention. At the time of this writing, the apE source is supported on a wide variety of equipment (see Figure 3).

Dependencies for optimization, debugging, and profiling are all transparent to developers, as are the mechanics of removing, compiling, and installing software on any system. The forge system was crucial to our completion of the software and will remain vital as we enter the support stage. For maximum portability, the entire forge system is implemented using C-shell syntax, so no compilation of these tools is necessary. Actual source code is stored using SCCS.[11]

No embedded graphics library

We chose not to embed any graphics library in the basis of our system. We have been criticized, and perhaps somewhat correctly, for not building our software upon a graphics-software layer such as CORE, GKS, PHIGS, PHIGS+, PEX, or others. In late 1987, when we faced this decision, the number of competing standards was large, and no clear winner had emerged. None of the standards available then were really sufficient for scientific visualization. Indeed, we would argue today that this situation really hasn't changed. Constructing our software on such a platform would be a tacit endorsement of one of these standards and would require users to obtain the necessary licenses to actually program within apE. Most

workstation vendors do not currently ship a PHIGS product, for example, as no-cost, bundled software with their systems. If we linked apE with PHIGS or any other standard, additional cost would be incurred in purchasing, installing, and maintaining a graphics library in addition to apE. All you need to run apE is apE.

An even more restrictive problem comes from the fact that most of the functions performed by apE are not currently part of any graphics standard. The mapping stage of the visualization pipeline is not adequately addressed by any standards at this time. Therefore, the only real value of a graphics standard lies in the rendering stage. A later port of the system is likely to use PHIGS[12] as the basis for rendering, just as the current version uses custom software on non-graphics workstations and Silicon Graphics's GL[13] on those geometry engines that support it.

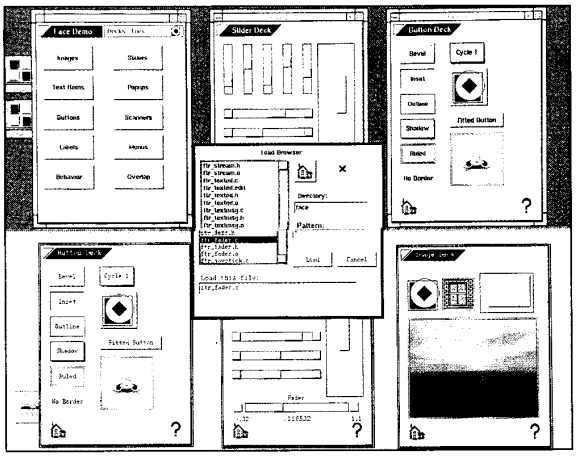


Figure 4. In this composite, the X11 version of face occupies the top half of the image while the Sunview version occupies the lower half.

New interface layer

We chose to build a new interface layer on top of existing "standards." Just as we have been criticized for not choosing a graphics standard, we also did not choose to build on one of the existing window systems. Clearly today the only "standard" window system is the X Window System[14]; in 1987, though, a number of pretenders to the throne threatened to steal the glory from X. Three years later, software and hardware developers alike still deserve better than the current state of the X Window System. Some vendors ship X11 Release 2; others ships X11 Release 3 or 4; still others are using variants of X10. Despite the claims, the intense battle between such competing higher level standards as Motif[15] and Open Look[16] will continue this uncertainty. Sun Microsystems's SunView continues to dominate as the window system of choice on their systems, and most applications written for Silicon Graphics machines are done either directly in their GL language or NeWS. Even today it simply is not the case that a user-level window interface standard exists, and it is unclear that the fog will lift in the near future.

Given this somewhat bleak outlook, we carefully examined the window systems on platforms we were likely to support, and came to the conclusion that three separate ports were required: SunView,[17] low-level X (at the toolkit level or below), and SGI's GL. Therefore, we designed an upper level, known as face, to serve as the standard "window system" for all apE software (see Figure 4). All interface implementation was done in the face library, thus making the "port" a matter of building the face layer on top of the three chosen window systems. At this writing, this work is primarily complete for X and SunView and is underway for SGI's GL.

This is not an ideal outcome in a large-scale software system development, since we have devoted nearly one third of our total person-power resources to the interface problem. The X Window System is far from solving the higher level problems of user interface. It is woefully complex, poorly documented, and exists in far too many vendor-customized versions. Portability is difficult or impossible to obtain above its lowest levels. It also suffers from a poor abstraction model of the workstation (too dependent on resolution) and countless inefficient implementations. Better interface systems, though, such as Sun's NeWS,[18] have been swept away in the self-aggrandizing rush to proclaim a standard. While the original concept of the X Window System may have been elegant, it now shows the unmistakable signs of design by committee and is a serious detriment to widespread interface development. The recent addition of Nextstep[19] to the fray and IBM's apparent endorsement of it further obfuscates the issue.

With source code control, interface, and (lack of) a graphics library in hand, we were ready to actually implement the application software. This phase of the development was divided into three logical elements: the construction of the libraries, the construction of the individual dataflow elements (or modules), and the construction of the tools and interface that would comprise the look and feel of apE to the average user. We began by implementing the library in phases and completing the Unix-level hiding functions first. The data language, user interface library, and graphics functions were done in preliminary test forms prior to

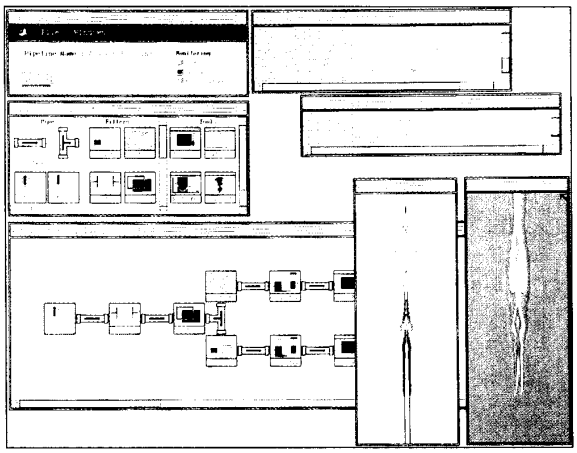


Figure 5. Pipeline construction and imagery created using apE 1.1.

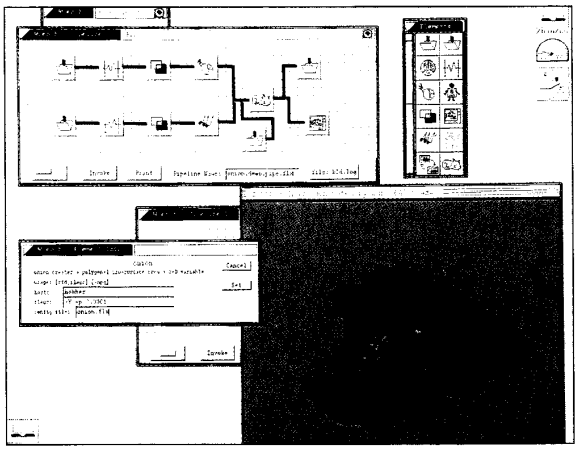


Figure 6. Pipeline construction and imagery under apE 2.0.

full implementation. The resulting test software was released (as version 1.1) and used to help motivate the full implementation of apE version 2.0.

Design and construction of apE 1.1

In the fall of 1988 the first version of the apE software was released for testing in the state of Ohio. In April 1989, the software was released to all Ohio Supercomputer Center users in what has become known as "version 1.1" (see Figure 5). Finally, in August 1989, this software was placed in the public domain and made available by tape or anonymous FTP for all academic and research users in the world. It remains available today from suna.osc.edu.

In addition to traditional tools for scientific visualization (such as contour and color plots), apE was unique in providing a flexible data language, a visual programming paradigm, a network-distributable execution syntax, and a generic user interface. While incomplete, these elements served as ideal test platforms for the concepts and goals set forth in the apE design.

The flow data format was implemented for use in apE 1.1. Binary independence was achieved by building on top of Sun Microsystems' XDR[20] data language. Initial flow elements included variables, grids, images, maps, and various control elements. Both C and Fortran bindings were included.

As a dataflow system, we felt it was very important that the user program it not by typing but by dragging iconic representations of operations onto a drawing grid and connecting them into the desired pipelines. This technique, recently popularized by AVS, was a part of the first apE release in early 1989.

The individual elements that comprise a pipeline in apE 1.1 all execute as separate processes in a data-driven mode and operate (to the extent possible) in parallel. They can be distributed to any local workstation (or to the Cray) by simply changing the name of the execution host for any desired module.

As a testbed for the face library concepts, a generic user interface was developed under SunView known as *sv_util*. This gave the apE system an entirely different look and feel from any commercial products. New buttons, sliders, scanners, alerts, and browsers made for a clean, functional interface that was replicated throughout all the tools distributed with apE 1.1.

While the visualization capabilities of apE 1.1 were limited to two dimensions, it remains a popular package, and has been used extensively at the Ohio Supercomputer Center.[21,22] In addition to the graphical programming tools, apE 1.1 includes contouring, color, and carpet plotting, as well as various data manipulation utilities. Time variant or multiframe data can be played back in flip-book style on a Sun workstation. A histogram tool provides a direct interactive interface for the normalization and preparation of scientific data. FORTRAN linkages allow direct connection of graphical pipelines to simulations, executing locally or on a remote supercomputer.

Due to a lack of contrast between text and background, this page did not reproduce well.

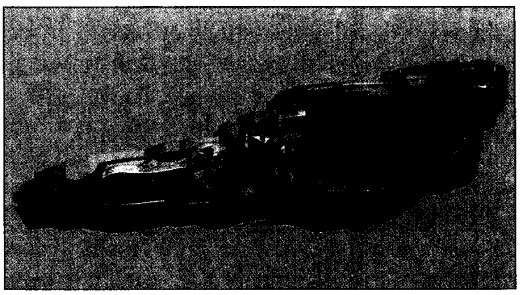


Figure 7. Iso-surface detection and rendering of Lake Erie's temperature (apE 2.0).

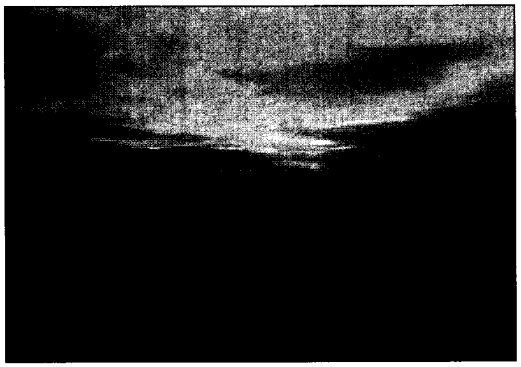


Figure 8. Realistic rendering of Lake Erie (apE 2.0).

Design and construction of apE 2.0

The lessons of apE 1.1 were used to improve the software iteratively for the second release, yet in many ways apE 2.0 represents the culmination of the concepts and ideas presented in this article (see Figure 6). The first release of the apE software taught its creators valuable lessons about software design, implementation, and distribution.

The data-language flow that had been developed for apE was enhanced, extended, and renamed *flux*. Specifically designed to deal with large amounts of data in user-designed grouping, flux is a powerful information management tool.[23] All data entities, from images to variables to pipeline descriptions to icons, are represented in *flux*. Even interfaces defined in the face system can be described in flux.

The generic user interface, first presented in apE as sv_util, was expanded and renamed *face*. The face libraries provide a complete, window-system-independent interface for program development. Face elements include most of the standard interface items, such as buttons, menus, sliders, scanners, and text-entry boxes. On top of this layer more complex elements are provided as well, such as alerts, browsers (for selecting a text element from a list), and collectors (for selecting several text elements from a list). Face provides a generic, application-based interface for interactive tool design that allows a single application to execute under SunView, X, and GL without significant source code changes.

The operational tools provided in the first release have also been significantly reworked. The pipeline construction tool has been reworked to increase interactivity and to handle different connection methods between the elements. (ApE 1.1 used Unix pipes to connect the dataflow elements; apE 2.0 uses both Unix pipes and sockets for connections.) A central console provides an outlet for error messages and access to documentation. An interactive image viewer allows manipulation of single and multiple images and real-time "playback" of image sequences. A geometry viewer allows interactive viewing of geometries.

While apE 1.1 was limited to nearly linear pipelines, apE 2.0 is designed to allow complex pipeline configurations, including multiinput, multioutput, and cyclic graphics. This cyclic capability provides apE 2.0 users with the ability to investigate connections between graphics and supercomputer simulations, and to attempt to "steer" a simulation through visual feedback. These additions are all natural extensions of apE 1.1.

Finally, the filters/modules have been extended to include 3D elements as well as the traditional 2D ones found in the first release of apE. Visualization techniques include carpet and contour plots, surface detection, terrain generation, and all forms of rendering from scan-line polygonal techniques[24] (see Figures 7 and 8) to ray tracing. A volumetric rendering system based on methods developed by Levoy[25] (see Figure 9) is also included. Particle tracing, advection, and surface feature detection (such as stream lines) are also included. In addition, full prototypes are provided to allow extension of the system by the addition of new filters, data types, tools, and interface elements.

Distribution philosophy

One of the real keys to the current and future success of the apE software effort has been the distribution policy. While a corporation must be concerned about

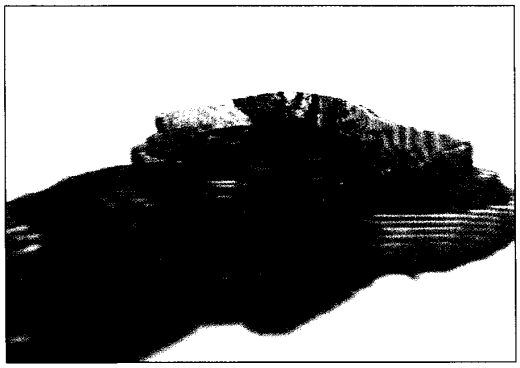


Figure 9. Volumetric rendering of Lake Erie's temperature distribution (apE 2.0).

profits, competition, market analysis, and other factors, we were able to concentrate solely on providing the best tools for the research community, knowing that our success would be judged by the productivity of our users, not the corporate bottom line. The best and only result we hoped for was widespread usage and increased productivity among Ohio's researchers.

The first version of apE was released in binary form only. For many of our users, this was insufficient, because it prevented them from using the software fully. Many people needed to modify the code to suit particular needs or demands in a particular application or field of interest. Some needed to make changes to suit local equipment or configurations. Finally, for many, not being able to see the source code caused a lack of confidence in the final results. Even if the code is not modified, it is of great value to examine sections to understand how a particular function is implemented or why an unexpected result is seen. University environments typically enjoy source code for most applications for precisely this reason.

We are now able to distribute the second version of the software in source code form. All of the apE system, including window-system layers, program-development layers, data-format layers, and all existing filters and tools will be released in source code form with the software. Academic and nonprofit institutions can request this software (with manuals) for a nominal duplication fee.

Advantages of academic software development

Clearly, many of the decisions and techniques described here are appropriate only because the developers are a part of an academic institution. Normal commercial firms cannot respond to market demands in the way we did because they are driven by corporate-profit responsibility, not an altruistic (and perhaps unrealistic) desire to change the way science is done. We were and are very lucky that our funding sources are predisposed to support this kind of large-scale development, and to make the final results widely available.

In many ways, though, we believe our success is indicative of the advantages that academic software development enjoys over commercial efforts. While traditionally academic software is poorly documented and distributed "as is," it is possible to build a high quality product within the university environment. The support we have received at Ohio State University and the Ohio Supercomputer Center has enabled us to make this software available widely—and cheaply.

Conclusion

The apE system does not represent a breakthrough in computer graphics. Most of the technology that has been harnessed to construct apE has been in existence for a number of years, and precious little of it could in any way be considered to be state of the art. However, the apE system does represent a significant new step in placing sophisticated tools into the hands of users. At the same time, our work and our distribution policy has helped to push industry toward a greater realization of the nature of the scientific visualization problem. The potential of visual methods for data analysis is enormous, and we need to recognize that the grand challenge that faces us today is not in making faster silicon but in finding new ways to improve the productivity of our research community. ■

Acknowledgments

I thank Charles Csuri, director of the Advanced Computing Center for the Arts and Design, who marshalled the resources to bring this project into being, and Charles Bender, director of the Ohio Supercomputer Center, who provided continuing support as the project grew. OSGP owes a great debt to these individuals, without whom this project would not have been possible.

I would also like to acknowledge some of our dedicated users, who have helped us bring apE from concept to reality. G. Comer Duncan of Bowling Green State University was instrumental in debugging the first release, and has been a source of inspiration

throughout the project. Keith Bedford of Ohio State University supplied the Lake Erie data. Shoichiro Nakamura supplied the gas-flame data. Many other Ohio Supercomputer Center users deserve our thanks.

This work was supported by the Ohio Board of Regents, through the Ohio Supercomputer Center, and by Ohio State University. This work was supported in part by a grant from Cray Research, by an equipment grant from Apple Computer, and by an equipment loan from Silicon Graphics.

References

1. B.H. McCormick, T.A. Defanti, and M.D. Brown, "Visualization in Scientific Computing," *Computer Graphics*, Vol. 21, No. 6, Nov. 1987.
2. C. Upson et al.,"The Application Visualization System: A Computational Environment for Scientific Visualization," *CG&A*, Vol. 9, No. 4, July 1989.
3. A. Giacalone et al., "VERITAS : Visualization Environment Research in the Applied Sciences," Proc. 1989 SPIE Conf. on 3D Visualization and Display Technologies, SPIE, Bellingham, Wash., 1989, pp. 127-134.
4. F.J. Bitz and N.J. Zabusky, "David and Visiometrics," to be published in *Computers and Physics*, July 1990.
5. *Precision Visuals' Workstation Analysis and Visualization Environment*, Precision Visuals, Boulder, Col., 1988.
6. H.S. Anderson, "Distributed Supercomputer Graphics Using Unix Tools," *Proc. USENIX Workshop on Unix and Supercomputing*, Boston, Sept. 1988, pp. 25-32.
7. R. Marshall, J. Kempf, S. Dyer, C-C Yen, "Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie," *1990 Symp. on Interactive 3D Graphics, Computer Graphics*,Vol. 24, No. 2, March 1990.
8. G. Kawasaki, *The Macintosh Way*, Scott, Foresman and Company, Glenview, Ill., 1990.
9. *Objective—C Compiler Version 4.0*, The Stepstone Corp., Sandy Hook, Conn., 1988.
10. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading, Mass., 1986.
11. *Project Ingres, the University of California at Berkeley, An Introduction to the Source Code Control System*, University of California at Berkeley, Calif., 1980.
12. *Programmer's Hierarchical Interactive Graphics System (PHIGS)*, Draft Standard ISO dp9592-1:1987(E), International Standards Organization, Geneva, Oct. 1987.
13. *The Graphics Language Programming Guide*, Silicon Graphics, Mountain View, Calif., 1989.
14. R. Scheifler and J. Gettys, "The X Window System," *ACM Trans. on Graphics*, Vol 5, No 2, Apr. 1986, pp 79-109.
15. *HP Motif Developers Release*, First ed., Hewlett-Packard, Corvallis, Ore, 1989.
16. R. Probst, "OPEN LOOK Toolkits," *USENIX Computing Systems*, Vol 1, No 4, Autumn 1988, pp 76-86.
17. *SunView Programmer's Guide*, Revision A, Sun Microsystems, Mountain View, Calif., 1986.
18. *Sun NeWS Technical Overview*, Sun Microsystems, Mountain View, Calif., 1987.
19. *NeXT Technical Documentation I*. NeXT, Inc. Palo Alto, Calif., 1988.
20. *Network Programming Guide*, Revision A, Sun Microsystems, Mountain View, Calif., 1990.
21. G. C. Duncan, "Head-on Collision of a Black Hole and a Star," *Proc. Fourth Science and Engineering Symp.*, Cray Research, Minneapolis, Minn., October, 1988.
22. J.S.Hobgood and R.S. Cerveny, "Ice Age Hurricanes and Tropical Storms," *Nature*, 333, 1988, pp. 243-245.
23. J. Faust and S. Dyer, "An Effective Data Format for Scientific Visualization," *Proc.1990 SPIE Conf. Extracting Meaning from Complex Data*, SPIE, Bellingham, Wash., Feb. 1990.
24. S. Dyer, "Supercomputer Rendering," presented 1987 Convex Users Group Meeting, Richardson, Tex.
25. M. Levoy, "Volume Rendering: Display of Surfaces from Volume Data," *CG&A*, Vol 8, No 3, May 1988, pp. 29-37.

D. Scott Dyer is an associate director of the Ohio Supercomputer Center. He has directed the apE project since its inception in 1987. Before joining OSC, he was on the staff of Cranston/Csuri Productions and an associate director of the Computer Graphics Research Group at Ohio State University. His research interests include high-quality realistic rendering, distributed processing, and the construction of software environments. He is a member of ACM and IEEE Computer Society.

Dyer received a BS in Mathematics from Carnegie Mellon University in 1981.

Dyer can be contacted at The Ohio Supercomputer Center, 1224 Kinnear Road , Columbus, Ohio 43212. Information about the availability of the apE system can be obtained through Michelle Messenger, apE Project Coordinator, at the above address, fax 614-292-7168 or through electronic mail at michelle@rhett.osgp.osc.edu.