Case study: applying scientific visualization to parallel performance visualization

Steven T. Hackstadt* and Allen D. Malony

Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
{hacks,malony}@cs.uoregon.edu

ABSTRACT

The complexity of parallel programs make them more difficult to analyze for correctness and efficiency, in part because of the interactions between multiple processors and the volume of data that can be generated. Visualization often helps the programmer in these tasks. This paper focuses on the development of a new technique for constructing, evaluating, and modifying sophisticated, application-specific visualizations for parallel programs and performance data. While most existing tools offer predetermined sets of simple, two-dimensional graphical displays, this environment gives users a high degree of control over visualization development and use, including access to three-dimensional graphics, which remain relatively unexplored in this context. We have developed an environment that uses the IBM Visualization Data Explorer system to allow new visualizations to be prototyped rapidly, often taking only a few hours to construct totally new views of parallel performance trace data. Yet, access to a robust library of sophisticated graphical techniques is preserved. The burdensome task of explicitly programming the visualizations is completely avoided, and the iterative design, evaluation, and modification of new displays is greatly facilitated.

Keywords: parallel processing, scientific visualization, performance evaluation, prototyping, iterative design

1. INTRODUCTION

Performance visualization is the use of graphical display techniques for the visual analysis of performance data to improve the understanding of complex performance phenomena. The effectiveness of multi-dimensional information displays has been well-documented [15]. While the graphics of current performance visualization tools are predominantly confined to two-dimensional displays, we have developed methods for rapidly prototyping next-generation, multi-dimensional performance visualizations.

This paper proposes a design process for performance visualization development that greatly reduces programming overhead, facilitates rapid prototyping, and allows for effective iterative design and evaluation. By applying the tools of scientific visualization to parallel performance visualization, we have found that such next-generation displays can be prototyped, if not implemented, in existing data visualization software products using graphical techniques that physical scientists have used for several years now.

We proceed by motivating this research in Section 2. Next, we summarize related research in Section 3 and offer a description of the visualization package, Data Explorer, in Section 4. This is followed by a brief examination of the methodology (Section 5) and current implementation environment (Section 6). Section 7 presents some example visualizations, and Section 8 evaluates and summarizes our results.

2. MOTIVATION

The importance of graphical visualization in parallel performance evaluation has been established by several research tools (particularly the ParaGraph tool [2,3]). Such products offer the parallel programmer access to information and insights that might otherwise go unobserved. Whether a parallel program is executing on 4 or 4,000 processors, the ability to view performance data in an useful way often enables the programmer to find anomalous behavior within a program. Subtle changes in the way a program performs its computation, for example, may result in substantial improvements in performance. Without visualization techniques to aid in the discovery of such problems, performance improvements might never be found [2].

S. Hackstadt and A. Malony. *Case study: applying scientific visualization to parallel performance visualization*. IS&T/SPIE Symposium on Electronic Imaging: Science & Technology, Conference on Visual Data Exploration and Analysis, San Jose, CA, February, 1995.

Currently, developers of parallel performance visualization tools decide prior to implementation what set of visualizations may be useful to the most people. However, current research strongly suggests the need for a wide range of application-specific visualizations to augment a general-purpose set [14]. To date, this need has been difficult to fulfill because of the considerable overhead in creating, evaluating, and formalizing performance visualizations. Even in so-called "extensible" tools, such displays are not easily created since they typically require special programming skills [3]. Clearly, a development process that requires little overhead and programming would enable developers to generate application-specific displays quickly in response to user needs, as well as create and evaluate general-purpose visualizations.

For several years, scientists have used generalized data visualization software products that have most, or all, of the tedious graphics and data manipulation programming already done (although there is still a creative process involved in constructing scenarios for visualizing scientific data). Performance visualization developers, on the other hand, have heretofore chosen to redevelop dedicated graphics and data manipulation support from the ground up for each new tool, inadvertently reducing the type and variety of displays available to the user.

As parallel computing architectures, environments, languages, and applications continue to advance, performance visualization needs become more demanding. Most existing tools are limited to two-dimensional displays, offer little customization and display interaction, and have strict data formats. Three-dimensional visualization and advanced graphical techniques applied in scientific fields have opened up entirely new possibilities for researchers, and they stand to do the same for the field of performance visualization.

To determine whether the field of performance evaluation can benefit by such "next-generation visualizations" requires a means for rapidly prototyping and evaluating new displays. To apply the development process of existing performance visualization products to these new displays would be to start coding hundreds of three-dimensional graphics routines and interaction techniques, not to mention advanced data representation and manipulation capabilities. Many months later, a researcher might be in a position to begin prototyping and evaluating new visualizations. But the necessary changes to those displays and each successive step of iterative design may significantly delay the project.

As will be seen, we were successful in developing techniques for rapid prototyping of displays with minimal overhead and programming expense. The process avoids graphics programming, yet maintains access to numerous display styles and interaction techniques. In essence, developers are able to focus on the visualization design rather than the underlying implementations of data models and low-level graphical operations.

This research, while greatly facilitating the evaluation of new and existing visualizations, does not have evaluation as its goal at this point. Many visualizations will be presented in what follows. Any "evaluation" is ultimately aimed at the development techniques being explored, not the visualizations themselves.

3. RELATED WORK

Tools such as ParaGraph [2,3] have become popular because of their portability, scalable displays, and complete performance visualization environments. A recent summary of current visualization tools for parallel systems can be found in [6] where Kraemer and Stasko summarize the current state-of-the-art in visualization development for parallel systems in general (*i.e.*, parallel debugging, program visualization, and performance evaluation).

Stasko also explains the dire need for application-specific displays in the context of parallel program debugging [14]. Similarly, Pancake covers the importance of display customizability as it pertains to parallel debugging [9]. Roschelle [11] argues that meaningful visualizations are not necessarily those that are consistent with an expert's mental model. Rather, users should be able to experiment with visualizations and develop their own understanding of the data. Collectively, these researchers demonstrate the important roles that users must play in both the development and use of visualizations.

The use of prototyping tools has been established by systems such as Pablo [10] and Polka [14]. Pablo promotes itself as a performance tool prototyping environment that supports end-user applications, but it provides little support for new visualization prototyping. Polka can be used effectively for the rapid development of algorithmic animations but is primarily suited for sequential programs.

Rover [12,13] utilizes the AVS and Matlab visualization packages to generate performance displays for parallel programs. Her approach is similar to the design process proposed in this paper in that performance data is collected, transformed into a format readable by the software, and then displayed using that software. The tools were used to generate simple two-dimensional displays. Our research both formalizes the approach and extends the use of such products into the development of new, more complex visualizations.

4. OVERVIEW OF DATA EXPLORER

IBM Visualization Data Explorer is an advanced data visualization software package that accommodates both developers and end-users of visualizations [4,7]. Visualizations can be built by creating a visual, dataflow program in the graphical user interface or by using a scripting language. The end-user can control many aspects of a visualization through the interface, while more advanced users can create new visualizations by adding, removing, or changing modules within the current visual program.

Data Explorer (DX) is not unique in its capability to create sophisticated visualizations and to be integrated into our visualization development process, however several features did play an important role in this work. For example, the DX data model is extremely flexible and can handle everything from rendering very regular, mesh-connected structures to completely arbitrary sets of polygons. At its heart, the DX data model utilizes sets of positions, connections, and data associated with one or both of these sets. This simple foundation offers tremendous flexibility in what the model can represent.

In addition, Data Explorer supports a high degree of data and software reusability. Because DX data is self-describing, visual programs retain significant generality and can consequently create a wide class of visualizations. Furthermore, a single DX data file can easily be processed by multiple visual programs.

Finally, the numerous graphical techniques available in DX can be combined in hundreds of ways, provided that the data satisfies the requirements of the given techniques. Since the data is self-describing, DX modules can generally "do the right thing" for the data they receive. For example, a module which renders glyphs at a set of locations based on some data value will choose the appropriate graphical technique depending on the dimension of the data (*e.g.*, spheres for scalar values or vector arrows for three-dimensional data).

5. METHODOLOGY

The work described in this paper exists in the context of a high-level abstract model that is currently guiding our performance visualization research [8]. At the heart of this model is the claim that if performance visualization is to play a key role in the evaluation of parallel computers and programs, then it must be established on a formal foundation that relates abstract parallel performance behavior to visual representations (*i.e.*, a "visual performance abstraction"). The binding of a performance abstraction to a view abstraction, mapping performance object outputs to view object inputs, represents a performance visualization.

A performance abstraction is the representation of managed performance complexity (*i.e.*, desired performance data analysis) and the performance characteristics to be observed. A view abstraction is a representation of the desired visual form of the abstracted performance data, unconstrained by the limitations of the graphics environment. Together, these two descriptions can be used to produce performance visualization software by generating interfaces to available graphical programming libraries or existing data visualization systems. Using this methodology, new parallel performance visualization development environments can be created and studied. (For a more detailed description of this methodology, see [1] and [8].)

Our research is concerned with the application of this methodology in a specific implementation environment - the Data Explorer scientific visualization system. Realizing such a methodology in a specific environment is not always straightforward, and the correlation between abstraction and implementation may not be cleanly defined. Nonetheless, we have been able to implement the concepts of performance and view objects in the context of an existing data visualization system. As will be discussed in the next section, our process implements performance objects by structuring raw trace data, and view objects manifest themselves in the way a developer creates a visualization within Data Explorer.

6. IMPLEMENTATION

This section discusses one way that the methods described in the previous section realize themselves in a practical context. The visualization development process that has evolved from our work with Data Explorer is illustrated in Figure 1.

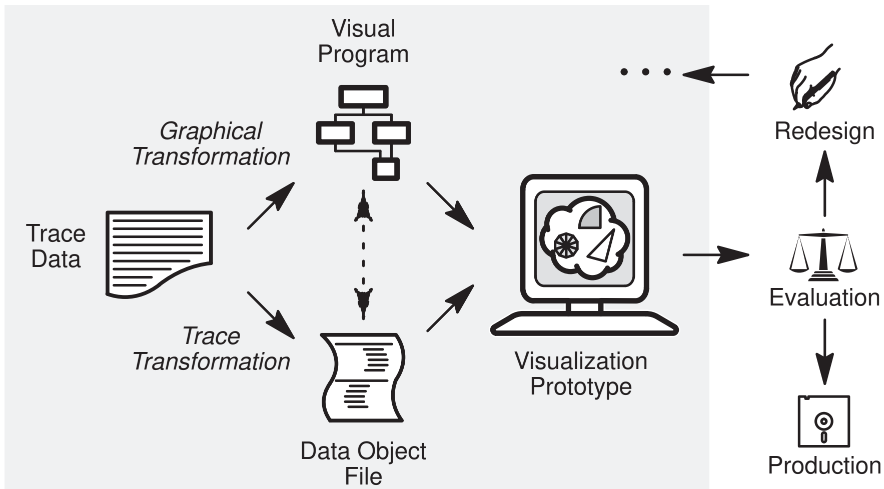

Figure 1. *Performance visualization is realized in the DX environment as transformations on trace data and graphical concepts.*

Our model for post-mortem performance visualization starts with raw trace data or statistics. The fundamental steps of the process transform the trace data into a data object file and a visual program. While trace transformations manifest themselves physically (*i.e.*, as some real program or function operating upon the trace data), graphical transformations in this environment are a mental process by which analysts merge the capabilities of the visualization environment, their knowledge of the performance data, and a visualization concept - or, more formally, a view abstraction - to construct a visual program in DX that will drive the creation of the desired display. The trace transformation may perform several operations and reductions on the data, but it ultimately creates a data object file that can be interpreted by the data visualization software. The content of these DX data object files maps nicely to the concept of performance objects. From this specially formatted data file, a visualization prototype is created by executing the visual program. The resulting display can be manipulated in many ways, including rotating, zooming, and travelling through or around the objects in the image by capitalizing on the capabilities available in the visualization software.

Figure 1 also shows how this process of prototyping performance visualizations can be integrated into an iterative design and evaluation process. Though not explicitly shown in the figure, the redesign of a visualization is accomplished by modifying the trace transformation, the graphical transformation, or both. If the software package is an inadequate environment for the final implementation (*e.g.*, because of cost or performance), then actual coding of the visualization is appropriate at that point. But this occurs only after the iterative design and evaluation process is complete. Ideally, though, the prototyping environment and implementation environment are the same, in which case the visualization prototype would be in its production version immediately after evaluation of the prototype is complete. The concept of a combined prototyping and production environment can also be found in Pablo [10].

In this process, graphics programming is avoided prior to production, the developer is able to focus on the visualization rather than the code that generates it, and visualizations are quickly created for evaluation, modification, and redesign. This is possible because changes to visualizations are made more easily and have fewer implications in the prototyping environment than in current performance visualization tools. Compared to existing performance visualization techniques, this method is different in that it separates the development phase from the production phase. As the area of performance visualization evaluation advances, a decoupled development process will be important so that modifications may be made quickly and easily and have limited side-effects. The application of our methodology creates a process that is a step toward that goal. (A more detailed examination of our implementation is available in [1].)

7. VISUALIZATION EXAMPLES

This section offers a few examples of visualizations that we have developed with Data Explorer. The first examples illustrate how existing performance visualization concepts can be extended beyond two dimensions. The second examples attempt to employ "sophisticated" graphical techniques and create displays that have never been used in the context of parallel performance and program evaluation.

7.1 Extending two-dimensional performance visualizations

This section explores the notion of extending traditionally two-dimensional visualizations into multiple dimensions. As an example of this, consider the popular Kiviat diagram [5]. The traditional form of this two-dimensional display has several spokes extending from a center point whose lengths change as time passes. A spoke corresponds to a single member of an object-set over which a scalar parameter (or a set of parameters) is being measured. As it has been applied in ParaGraph [3], a spoke's length represents the performance of a processor in a parallel computer. Then, when the ends of adjacent spokes are connected, triangular regions result. If the quantity being represented by the length of the spokes is, say, computation, then processor utilization would be "good" when the spokes are longest. (Spoke-length is typically mapped onto the interval [0,1]. Thus, with a large number of spokes at maximum length, the Kiviat diagram approximates a unit circle, which is sometimes used as a backdrop.) Figure 2 shows a single frame of an animated visualization created with Data Explorer.

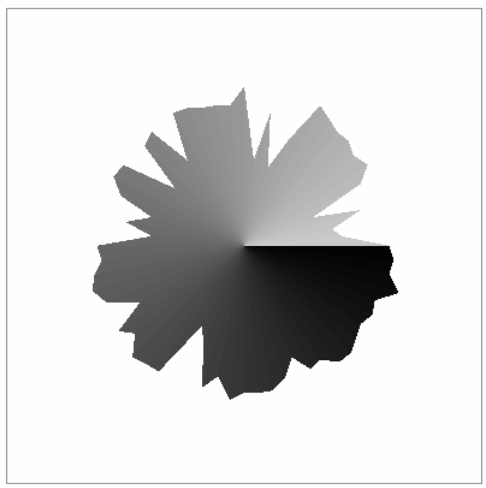

Figure 2. *A prototype of a traditional two-dimensional Kiviat diagram is easily reproduced in a scientific visualization package.*

One of the potential problems with a standard Kiviat animation is that the viewer sees only one step at a time and can easily lose track of how the performance at that step compares to the performance during the rest of the animation. Thus, by removing the animation of the display and letting time run along the third axis, a Kiviat "tube" results.

To create a tube with a solid exterior shell, the quadrilateral surface patches formed between corresponding processors in subsequent time steps are rendered instead of the triangular sections emanating from a slice's center (as in the 2-D Kiviat display). Still, the transformation required is only slightly more complicated than that for the standard Kiviat. Figure 3 shows a Kiviat tube generated by Data Explorer.

This representation of the original Kiviat diagram is important because it gives the viewer a very global view of the performance data, as opposed to the standard two-dimensional version which limits the viewer's ability to compare the performance of the application at different times during the trace. However, the three-dimensional representation tends to obscure more detailed information about individual processors at specific times, whereas the standard Kiviat display shows that information more clearly.

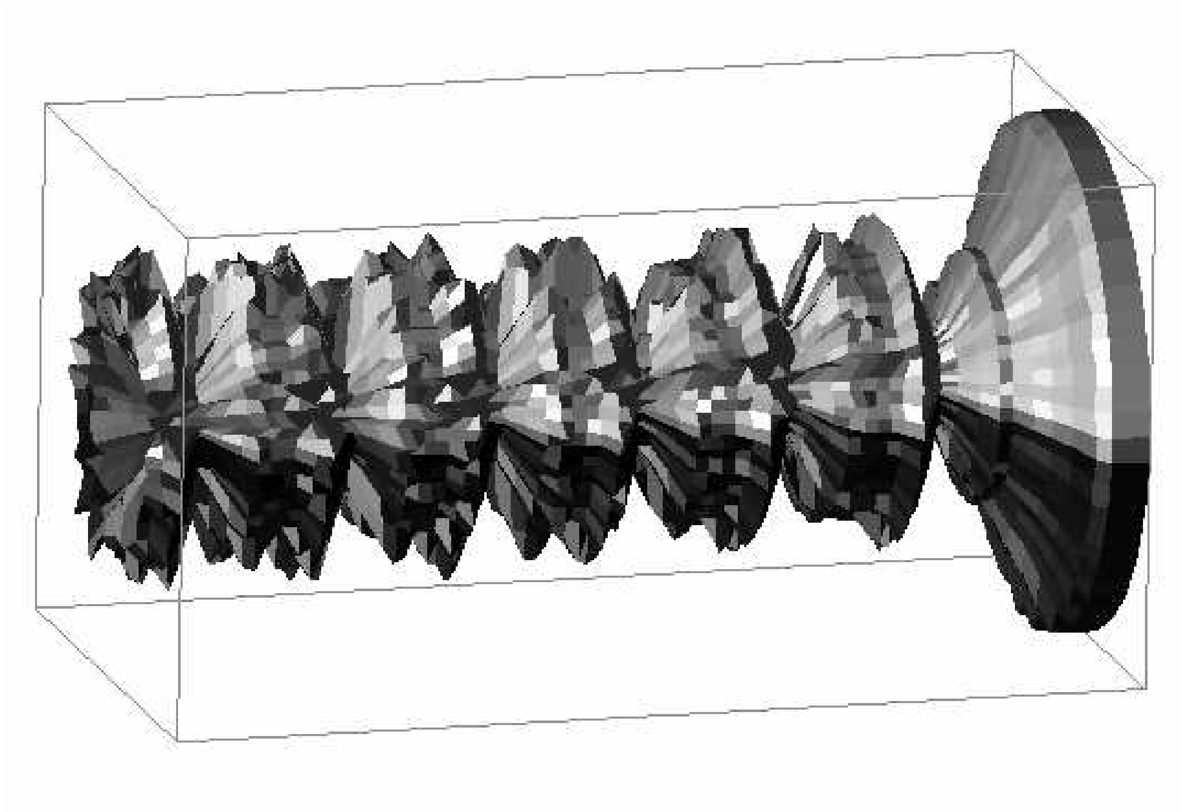

Figure 3. *A three-dimensional Kiviat tube generated by Data Explorer gives the viewer a more global perspective of the performance data, but at the same time obscures detailed information about individual processors or time steps.*

Some of Data Explorer's true power is revealed in the following example. It is possible that individually, neither the two-dimensional Kiviat diagram or the three-dimensional display just presented totally fulfills the viewer's needs. The two-dimensional version allows the viewer to assess how processors relate to each other during a given time slice, but makes it difficult to see how performance in one time step relates to other parts of the animation. The three-dimensional display tends to do just the reverse; that is, seeing trends over the life of the trace is easier, but it is difficult to see how processors relate to each other during a given time step. It may be that by combining the two displays, both needs could be met. Thus, the idea for an enhanced display is to let the two-dimensional Kiviat "slice" pass through the Kiviat tube, which can be made partially transparent. The slice highlights the interprocessor relationships for a given time step while the rest of the tube still reveals how a particular step relates to the rest of the data. The display is animated by letting the slice slide through the tube. Alternatively, the viewer can directly specify the time step at which to place the slice.

This is a complex visualization concept that combines several graphical techniques. However, having previously specified the two pieces of the display individually, Data Explorer allows the developer to combine the two trivially. In what literally took just minutes, the composite visualization in Figure 4 was created.

7.2 Using sophisticated graphical techniques

In most cases, visualizations are created by starting with an idea for a display and then figuring out how that could be accomplished using the visualization tool. However, developing new performance visualizations with visualization packages such as Data Explorer can lead to the creation of totally new performance displays thanks to the presence of many pre-programmed graphical techniques. Exposure to a new graphical realization technique can give developers ideas for performance displays that might not have been conceived otherwise. Of course, the value of any new visualization is unknown until it is thoroughly evaluated; this is true regardless of how the visualization was created.

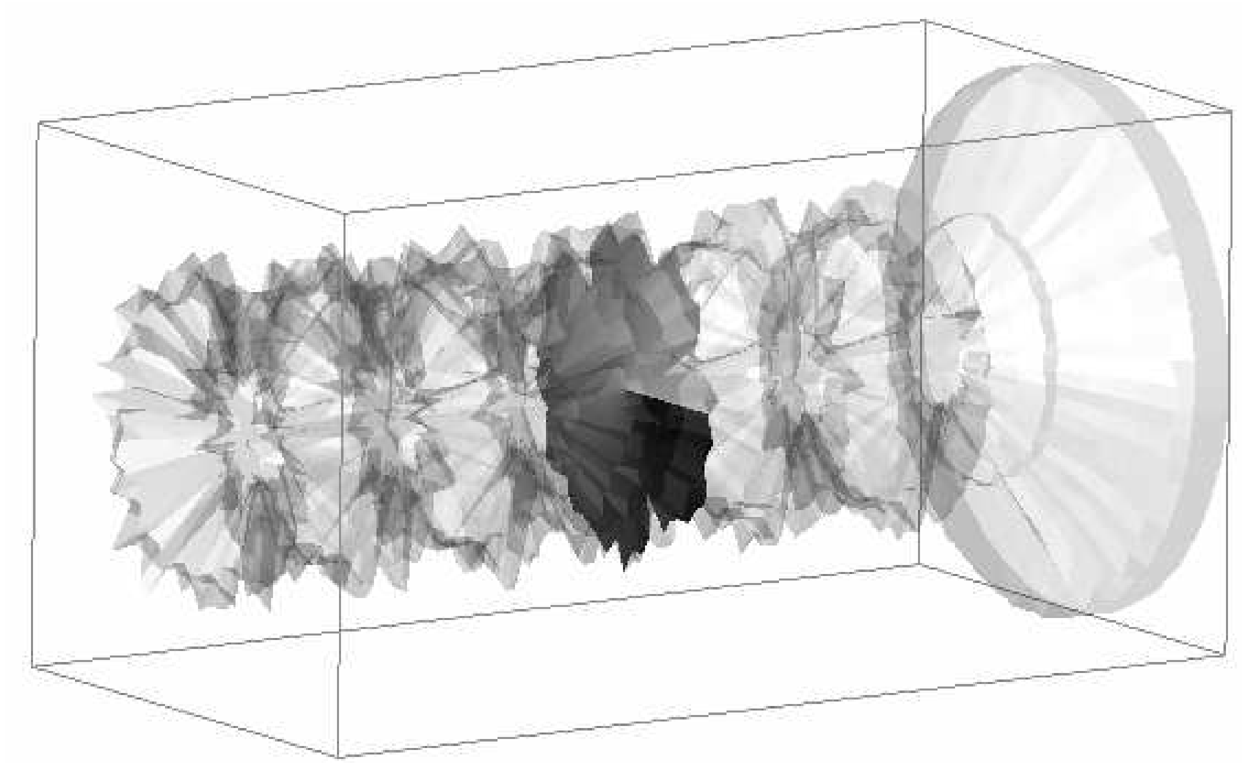

Figure 4. *A visualization which combines the traditional Kiviat display in Figure 2 and the three-dimensional one shown in Figure 3 utilizes several advanced graphical techniques, but by reusing the two simpler displays, Data Explorer allows the user to combine them effortlessly.*

Data Explorer has the capability to realize data using a technique called a *rubber sheet*. The concept is simple: a grid of positions and connections is interpolated to form a continuous "sheet"; the data values associated with each position are then used to displace (and color) that position on the sheet a distance proportional to the value in a direction perpendicular to the sheet. The result is a grid that is distorted (and colored) to reflect the data values of the grid positions.

Thus, in examining this graphical realization technique, the idea for a visualization evolved. The visualization's goal was to graphically represent the difference between local and remote accesses made by the processors of a parallel machine to the distributed data elements of a parallel application. Having developed the visualization's concept from a Data Explorer technique, all that remained was to create the trace transformation necessary to realize the visualization. Figure 5 contains several frames of the animation of this visualization.

In a manner similar to the example above, the visualization in Figure 6 was developed using a Data Explorer realization technique called an *isosurface*. Isosurfaces manifest themselves as contour lines for two-dimensional data (like elevation lines on a topographic map) and contour surfaces for three-dimensional data. In DX, an isosurface represents a set of (actual and interpolated) data points of constant value (or, an *isovalue*). Such a technique provides an interesting view of larger quantities of performance data when represented in a three-dimensional, volumetric structure such as a cube. Using this visualization technique as a starting point, we designed the visualization seen in Figure 6. It consists of a set of 64 processors arranged in a 4x4x4 cube. The level of processor utilization was used to create and color the isosurfaces. Each frame of Figure 6 contains a set of isosurfaces for five isovalues distributed between 0 and 1 (the range of values covered by the metric). These surfaces give the viewer an interesting perspective on the performance of the parallel application. In addition to revealing the global level of performance, the visualization potentially offers insight to the scalability characteristics of the application.

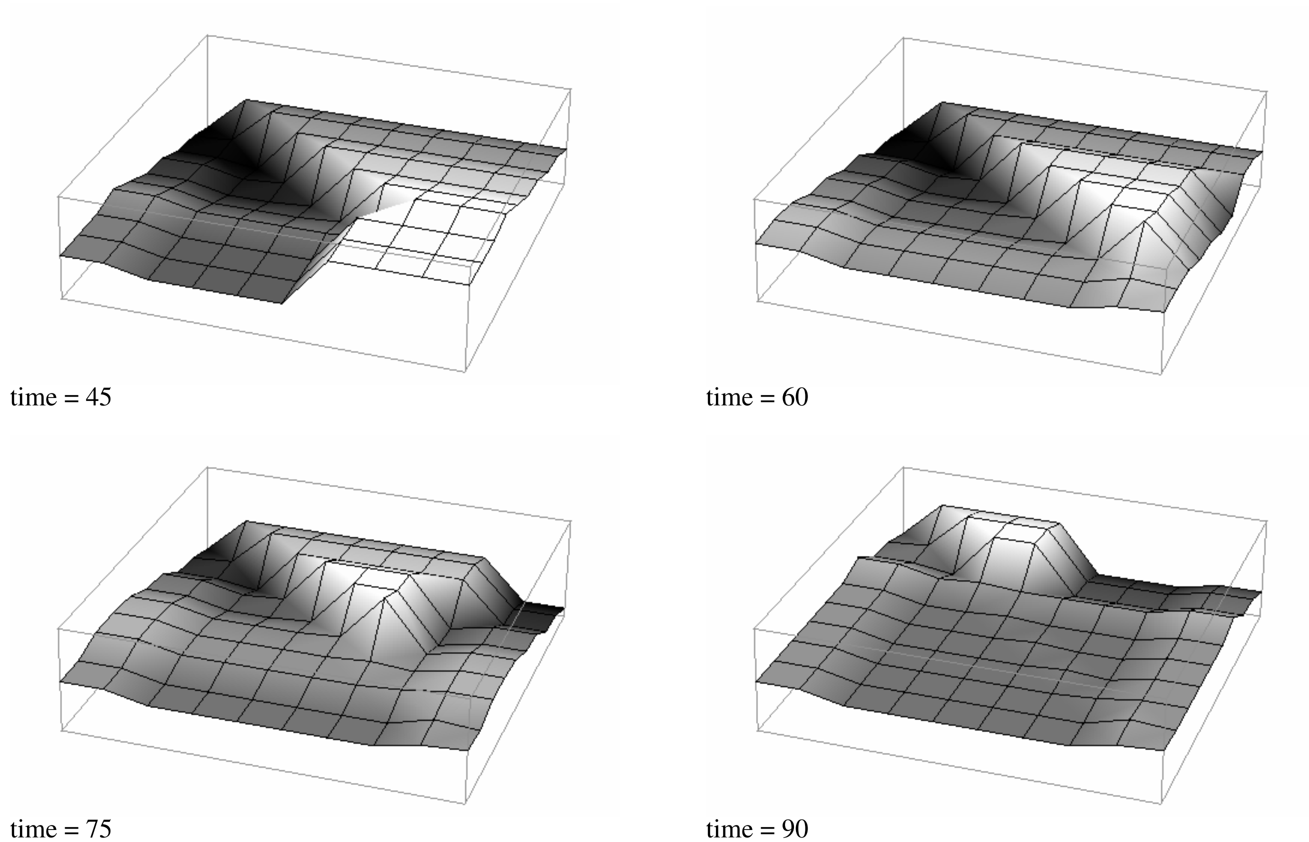

Figure 5. *This sequence of images partially illustrates Data Explorer's animation capabilities. These displays show the difference between local and remote accesses made to distributed data elements over the course of a Gaussian elimination algorithm. The higher the point, the more local accesses it has received.*

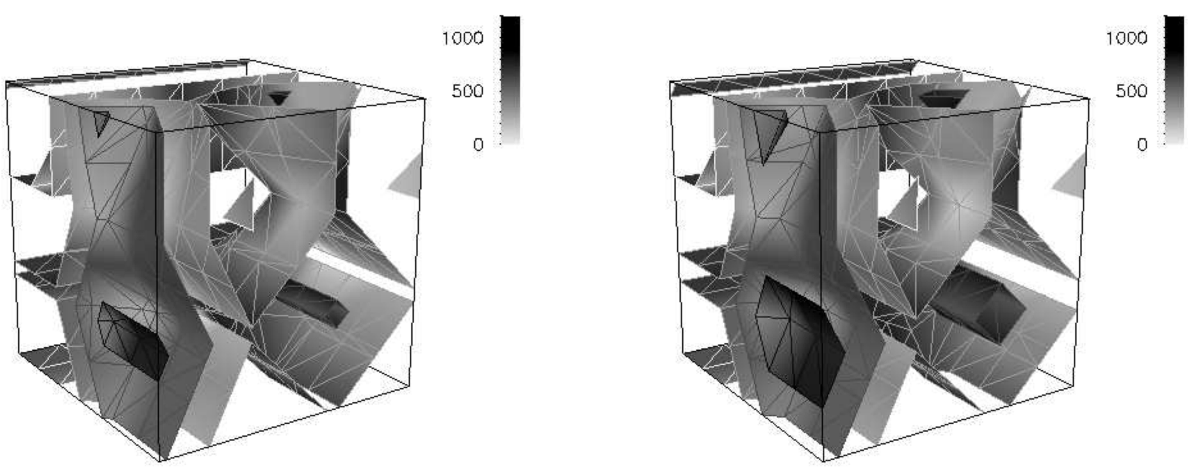

Figure 6. *Showing levels of constant use within a structure, isosurfaces are used to explore performance relationships among 64 processors arranged in a cube.*

8. CONCLUSION

The most obvious benefits of this approach to visualization come from the minimal overhead that developers incur by using existing software products. The primary overheads involved in the process are limited to the initial cost of the software and learning the visualization environment and underlying data model. Traditional visualization development has a single, overwhelming overhead: programming. Unfortunately, evaluating this new process must go beyond just the overheads that are encountered during development. In fact, if this type of implementation environment is used for prototyping, but the final implementation is to be built from scratch, then both overheads are incurred. Even in this worst case scenario, we feel that there are advantages to be gained by using the proposed methods.

It was mentioned earlier that our visualization method fosters iterative design and evaluation by minimizing the impact that modifications to either the data or graphical aspects of the visualization have on other parts of the visualization process. This is perhaps the most important and relevant aspect of this project to the field of performance visualization in general. Studies of performance visualizations have been few because of the effort required just to create the displays. Developers are forced to devote the majority of their time to building the visualization tool rather than evaluating visualization usability [8]. Formal evaluation is apparently sacrificed for two primary reasons. First, once a tool is completed, so much time has already gone into the project that developers can't afford to extend their work to include evaluation. Second, even if evaluations were done, modifications could impose additional significant programming costs on the project. Even those tools which market themselves with buzzwords like "modular" and "extensible" usually require considerable programming expertise [3]. Using software like Data Explorer stands to refine the field of performance visualization by enabling researchers to more easily conduct usability studies and perform formal evaluations of visualizations - to determine what displays are indeed useful.

On the other hand, it is possible that generalized data visualization packages contain too much functionality, resulting in "environment overkill." In other words, the software may be too general, leading to poor performance and unnecessary complexity. Our experience supports this to a limited degree. Our work with Data Explorer was done on an IBM RS/6000 Model 730 which required additional graphics hardware assistance and greater than 64 megabytes of memory to be acceptable. Tools built specifically for performance visualization may not have such stressing requirements on the computing system. But the hardware requirements made by scientific visualization packages are not without reason. These systems offer extremely flexible and powerful graphical techniques. For example, performance visualization developers would undoubtedly find the capability to compose a new display from two or more simpler displays advantageous, as was done in the Kiviat diagram examples discussed earlier. Furthermore, these products offer flexible data models that can be used to generate a wide range of visualizations and are designed to handle large, multi-dimensional data sets. Display interaction and animation are typically included, and the software is usually portable across many architectures.

In conclusion, this paper has proposed a design process by which performance visualization developers can drastically reduce the costly and time-consuming overhead of programming, yet gain significant power and flexibility in the displays they can generate. The application of a high-level visualization methodology to general data visualization software, like IBM's Data Explorer, leads to a design technology that improves the visualization development process by increasing both the quality of the displays and the speed with which they are created. The process lends itself to iterative design and evaluation, which is required to validate a display's usability - techniques that up until now have not been widely applied in the arena of performance visualization. Furthermore, the work being done here and at other research sites strongly suggests that users will demand interaction with and customization of next-generation performance displays. Existing data visualization software products are designed with these very capabilities in mind and can play a critical role in the development of visualizations for parallel performance and program analysis tools.

9. ACKNOWLEDGMENTS

This work was supported by an IBM Research and Development contract (MHVU3704) from the IBM Highly Parallel Supercomputing Systems Laboratory and a grant (ASC9213500) from the National Science Foundation, Advanced Scientific Computing Program.

This text and images of this paper, as well as many other research results, are available over the World Wide Web at URL *http://www.cs.uoregon.edu/~hacks/info-research.html*.

10. REFERENCES

[1] S. Hackstadt. *Prototyping Advanced Parallel Program and Performance Visualizations*. Master's Thesis, Department of Computer and Information Science, University of Oregon, June, 1994.

[2] M. Heath and J. Etheridge. *Recent Developments and Case Studies in Performance Visualization Using ParaGraph*. Proceedings from the Workshop on Performance Measurement and Visualization of Parallel Systems, Moravany, Czechoslovakia, October, 1992.

[3] M. Heath and J. Etheridge. *Visualizing the Performance of Parallel Programs*. IEEE Software, September, 1991, pp. 29-39.

[4] International Business Machines Corporation. *IBM Visualization Data Explorer User's Guide*, 2nd ed. August, 1992.

[5] K. Kolence and P. Kiviat. *Software Unit Profiles and Kiviat Figures*. ACM SIGMETRICS, Performance Evaluation Review, September, 1973, pp. 2-12.

[6] E. Kraemer and J. Stasko. *The Visualization of Parallel Systems: An Overview*. Journal of Parallel and Distributed Computing, 18, 1, June, 1993, pp. 105-117.

[7] B. Lucas, G. Abram, N. Collins, D. Epstein, D. Gresh, and K. McAuliffe. *An Architecture for a Scientific Visualization System*. Proceedings of Visualization '92, Boston, MA, October, 1992, pp. 107-114.

[8] A. Malony and E. Tick. *Parallel Performance Visualization*. Proposal to the National Science Foundation, CISE/ASC, Grant No. ASC 9213500, February, 1992.

[9] C. Pancake. *Customizable Portrayals of Program Structure*. Proceedings from the ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, CA, May, 1993, pp. 64-74.

[10] D. Reed, R. Aydt, T. Madhyastha, R. Noe, K. Shields, and B. Schwartz. *An Overview of the Pablo Performance Analysis Environment*. Univ. of Illinois Board of Trustees, November, 1992.

[11] J. Roschelle. *Designing for Conversations*. Paper presented at the AAAI Symposium on Knowledge-Based Environments for Learning and Teaching, Stanford, CA, 1990.

[12] D. Rover. *A Performance Visualization Paradigm for Data Parallel Computing*. Proceedings of the 25th Hawaii International Conference on System Sciences, 1992 (mini-conference on Parallel Programming Technology, Software Technology Track).

[13] D. Rover and A. Waheed. *Multiple-Domain Analysis Methods*. Proceedings from the ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, CA, May, 1993, pp. 53-63.

[14] J. Stasko and E. Kraemer. *A Methodology for Building Application-specific Visualizations of Parallel Programs*. Journal of Parallel and Distributed Computing, 18, 1, June, 1993, pp. 258-264.

[15] E. Tufte. *Envisioning Information*. Graphics Press, Chesire, CT, April, 1991.