Next-Generation Parallel Performance Visualization: A Prototyping Environment for Visualization Development

Steven T. Hackstadt and Allen D. Malony

{hacks,malony}@cs.uoregon.edu
Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
ph: 503-346-4407 / FAX: 503-346-5373

Abstract. A new design process for the development of parallel performance visualizations that uses existing scientific data visualization software to prototype new performance visualizations can lead to drastic reductions in the graphics and data manipulation programming overhead currently experienced by performance visualization developers. The process evolves from a formal methodology relating performance abstractions to visual representations under which performance visualizations are described as mappings from performance objects to view objects, independent of any graphical programming. This prototyping environment also facilitates iterative design and evaluation of new and existing displays. Our work examines how an existing data visualization tool can provide a robust prototyping environment for next-generation parallel performance visualizations.

1 Introduction

> Even though we navigate daily through a perceptual world of three spatial dimensions and reason occasionally about higher dimensional arenas with mathematical ease, the world portrayed on our information displays is caught up in the two-dimensionality of the endless flatlands of paper and video screen.... *Escaping this flatland is the essential task of envisioning information - for all the interesting worlds (physical, biological, imaginary, human) that we seek to understand are inevitably and happily multivariate in nature. Not flatlands.*
>
> \- Edward Tufte, *Envisioning Information*

Performance visualization is the use of graphical display techniques for the visual analysis of performance data to improve the understanding of complex performance phenomena. From the opening chapter of his book *Envisioning Information* [24], Edward Tufte foretells the future of performance visualization. Tufte recognizes and demonstrates the necessity and effectiveness of multi-dimensional information displays. While the graphics of current performance visualizations are predominantly confined to the two-dimensional flatland described by Tufte, our work has as one of its primary goals the development of new methods for rapidly prototyping a new generation of 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 performance visualization, we have found that 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.

Section 2 motivates the need for new development techniques in performance visualization that utilize advanced graphics, user interaction, customizable displays, and other characteristics. A description of related work is given in Section 3. This is followed by a high-level, abstract description of performance visualization in Section 4. Section 5, which presents several examples of new visualizations, is followed by a discussion of the proposed design technology in Section 6.

2 Motivation

By now, the importance of graphical visualization in parallel performance evaluation has been established by tools such as ParaGraph [6,7], Sieve [21], Pablo [17], Voyeur [22], Seeplex [2], and Traceview [13]. These tools offer the parallel programmer access to visual information and insights that might otherwise go unobserved. Subtle changes in the way a program performs its computation, for example, could offer substantial improvements in performance. Without visualization techniques to aid in the discovery of such problems, performance improvements might never be found [6].

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 (plus advanced graphical techniques) applied in the physical sciences has opened up entirely new possibilities for researchers, and it stands to do the same for performance visualization. However, performance visualization developers have heretofore chosen to develop dedicated graphics and data manipulation support from the ground up, inadvertently reducing the type and variety of displays available to the user.

To determine whether the field of performance evaluation can benefit by "next-generation visualizations" requires a means for rapidly prototyping and evaluating such new, complex 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.

We propose a methodology based on a formal foundation in which performance abstractions are mapped to visual abstractions [14]. In general, the methodology functions as an interface to existing visualization systems, programming libraries, and other graphics resources. Thus, an existing data visualization package is but a single means of implementing the formal, high-level abstractions. While our work has focused on the use of IBM's Data Explorer (DX) [9,12], any number of similar products could also be applied.

We have successfully prototyped performance visualizations in three categories:

- *Existing two-dimensional performance visualizations,*
- *Enhanced three-dimensional versions of existing visualizations, and*
- *New multi-dimensional performance visualizations that include advanced graphical rendering techniques.*

Our experiences in each category offer insight to the advantages of this approach. As will be seen, we were successful in developing techniques for rapid prototyping of displays with minimal overhead and programming expense, making displays available for evaluation without committing months of programming to the project. In fact, the process avoids graphics programming completely, yet it maintains access to numerous display styles and interaction techniques. In essence, developers are able to focus on the visualization design rather than the underlying implementation 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 present goal. Many visualizations will be presented in the pages that follow. Any evaluation that we do is ultimately aimed at the development techniques being explored, not the visualizations themselves.

3 Related Work

Tools such as ParaGraph [6,7], Pablo [17], Seeplex [2], and many others have become known for their portability, scalable displays, and complete performance visualization environments. A recent summary of visualization tools for parallel systems can be found in [11] by Kraemer and Stasko. The displays offered by these products can be very effective for general parallel performance evaluation, but do not effectively facilitate application-specific displays [6].

Many of the philosophies underlying our research are echoed in the work by Sarukkai and Gannon [21]. They call for application-specific displays, rapid prototyping techniques that facilitate usability studies, display interaction, and user control. An existing data visualization tool coupled with the design process proposed herein can provide many of these capabilities.

Stasko [23] also explains the dire need for application-specific displays in the context of parallel program debugging, but he differentiates parallel debugging from performance evaluation by claiming that performance visualizations do not focus on the semantics of a particular program. However, this research suggests that performance evaluation can be enhanced by creating visualizations which *are* linked to the semantics of an application. In this way, the visualization concerns of parallel program debugging and performance evaluation do intersect.

The use of prototyping tools has been established by systems such as Pablo [17] and Polka [23]. Pablo promotes itself as a performance tool prototyping environment that supports end-user applications, but provides little support for new visualization prototyping. Polka can be used more effectively for the rapid development of algorithmic animations, but is primarily suited for sequential programs.

Pancake’s discussion of display customizability, as it pertains to parallel debugging [16], claims that visualizations based on the user's conceptual model can be more meaningful than those which are not. Therefore, giving the user the ability to customize and/or control visualizations should result in more meaningful displays. Roschelle [18] 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.

Rover [19] calls for the use of analysis techniques used in traditional data visualization, including animation, image transformations, color manipulations, and statistical analyses. In [20] Rover utilizes the AVS and Matlab scientific visualization tools to generate performance displays. The tools were used to generate simple two-dimensional displays using an approach similar to the one proposed in this paper. Our research both formalizes the approach and extends the use of such tools into the development of new, more complex visualizations.

4 Methodology

The work described in this paper exists in the context of a high-level abstract model that is currently guiding the performance visualization research at the University of Oregon [14]. 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 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 a representation of managed performance complexity (*i.e.*, desired performance data analysis) and 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 developed and studied.

Figure 1 illustrates the methodology. Through a process of abstraction, performance data generates specifications for the performance analysis and the visual representation of the data. In practice, these abstractions would best be created through a specification language (one of which is currently being developed by our research group). Our particular implementation environment, Data Explorer, does not currently offer support for this aspect of the methodology. Specifications are instantiated to implementable objects. The term “objects” is intentionally general to accommodate the many realizations possible in different implementation environments. Figure 1 indicates an overlap between performance and view objects suggesting an interdependence between their implementations. Our research has found that the degree to which these objects can be implemented independent of one another is determined by the environment in which the implementation is taking place. In the ideal implementation, performance and view objects would be totally independent. In Data Explorer, this overlap is present in varying degrees, depending on the particular visualization. Next, the instantiated objects are combined to create the visual representation of the performance data. Finally, the actual rendering of the visualization takes place through a toolkit which interfaces with the various graphics resources being utilized. The explicit existence of the toolkit is also dependent on the implementation environment. In Data Explorer, the toolkit interface is embedded within the visualization system itself. For a more detailed description of this methodology, see [14].

The research documented in this paper is concerned with the application of this methodology in a specific implementation environment - a scientific visualization system; in this case, that system is Data Explorer. As mentioned above, realizing this methodology in a specific environment is not always straightforward, and the correlation between abstraction and implementation is not always cleanly defined. It was mentioned above that Data Explorer does not offer support for the specification of abstractions, and consequently, this was not the focus of our research effort. On the other hand, we have been able to implement the concepts of performance and view objects in the context of an existing data visualization system. The process that has evolved implements performance objects by structuring raw trace data so that the visualization system can process it. View objects manifest themselves in the way a developer creates a visualization within Data Explorer. For a more detailed examination of how performance and view objects are manifested in Data Explorer, consult the full technical report version of this paper [4].

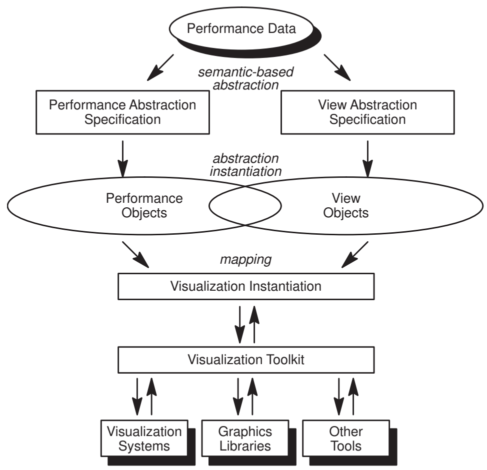


Figure 1. A high-level, abstract view of performance visualization.

5 Application

This section will examine several examples of how the design process has been applied to the performance visualization research being done at the University of Oregon. The three primary categories of visualizations presented earlier will be discussed, and each will offer insight into this new performance visualization development model. The reader should note that the greyscale renderings in this paper do not do justice to the full-color displays they represent.

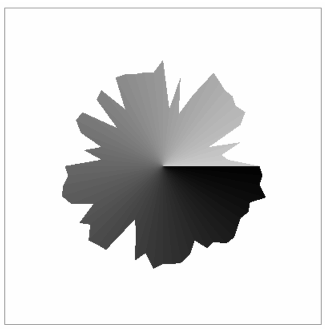
5.1 Existing Visualizations

By prototyping existing displays, we establish that the proposed design process can accomplish many of the same tasks as existing visualization tools. As an example of this, consider the popular Kiviat diagram [10]. The traditional form of this two-dimensional display has several variable-length spokes extending from a center point. A spoke corresponds to a single member of a set of objects over which a scalar parameter (or a set of parameters) is measured. As it has been applied in ParaGraph [7], a spoke represents a processor in a parallel computer, and the measured quantity is the level (*i.e.*, percentage) of computation for that processor during the last time interval. When the ends of adjacent spokes are connected, an overall shape results. 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. Figure 2 shows a single frame of an animated visualization created with Data Explorer. The importance of this approach becomes evident when one realizes that a visualization concept can be transformed into a prototype display in just a few hours.

5.2 Enhancing Existing Visualizations

Having successfully prototyped an existing display, the next step is to see how we can extend or enhance such visualizations. In this section, we continue our examination of the Kiviat display.

Our visualization group has experimented with three-dimensional representations of existing two-dimensional displays, among other parallel performance visualizations [14]. 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 one 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. Figure 3 shows such a display that was generated by Data Explorer. This alternate representation of the original Kiviat diagram is important because it gives the viewer a global perspective 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.



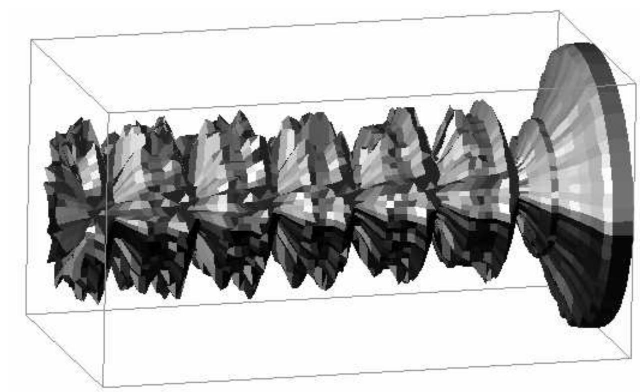


Figure 2. (left) A prototype of a traditional two-dimensional Kiviat diagram is easily reproduced in a scientific visualization package.

Figure 3. (right) A three-dimensional Kiviat tube gives the viewer a more global perspective of the performance data, but at the same time obscures detailed information about certain processors or time steps.

It is possible that individually, neither of the Kiviat displays generated thus far (Figures 2 and 3) totally fulfills the viewer's needs. The two-dimensional display 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 somehow, 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 via the scientific visualization system. 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 can be 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 created the two pieces of the display individually, the visualization software allows the developer to combine the two trivially. In what literally took just minutes, the composite visualization in Figure 4 was created.

5.3 New Visualizations

New visualizations represent the last category of displays that we have prototyped with Data Explorer. The reader should note that the presentation of these new displays is meant to illustrate the usefulness of this particular visualization technique as opposed to that of the visualizations themselves.

Traditionally, visualizations have been developed out of a dire need to see data presented in a certain way, but our earlier motivation of providing visualization techniques that better accommodate the rapid generation of *new* displays supports another, potentially more powerful method of visualization development where a feature or graphical technique available in the software motivates the concept for a performance visualization. This method can stimulate creative ideas that might not otherwise be conceived, and for the developer looking to create new and novel visualizations, these ideas may be helpful.

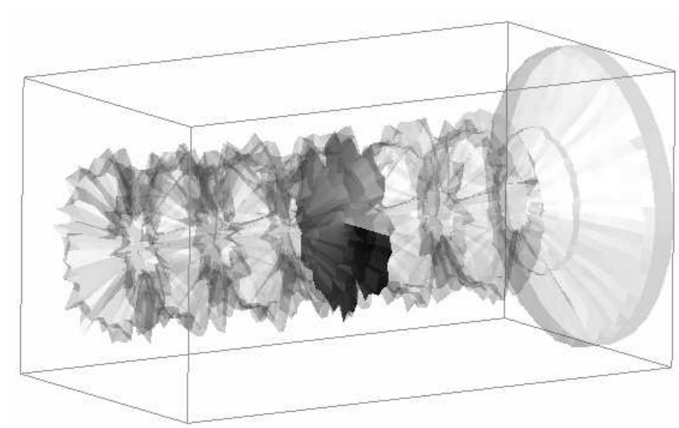


Figure 4. A combination of the Kiviat displays shown in Figures 2 and 3, this display utilizes several advanced graphical techniques. The visualization software allows the user to compose the two simpler displays effortlessly.

Working from Visualization Concept to Visualization Technique. 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. The first example is a three-dimensional visualization which reflects the state of a processor with three commonly traced metrics: percentages of computation, overhead, and idle times during the previous time interval. The concept behind the visualization is to represent each processor as a sphere (or glyph) in space. The location of each sphere is determined by the values of the three metrics corresponding to each processor. Thus, the axes correspond to computation, overhead, and idle. As time passes, the spheres move around the space [23]. Figure 5 contains an example of this visualization.

As with the other examples, it took less than a day to develop the basic prototype for this display. After that, Data Explorer's flexibility allows the user to customize and "tweak" the display to no end. The user has simple control over the size of the glyphs, animation speed and granularity, colors, and other features that are typically fixed in most performance visualization tools. These types of interactions are available directly from the scientific visualization package and do not require new transformations of the data or additional programming.

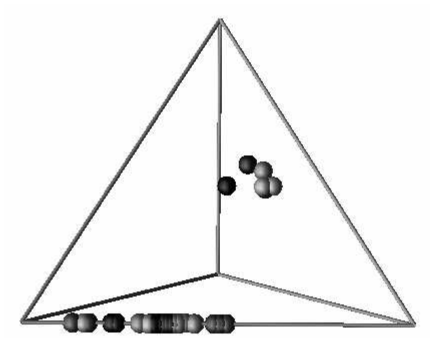


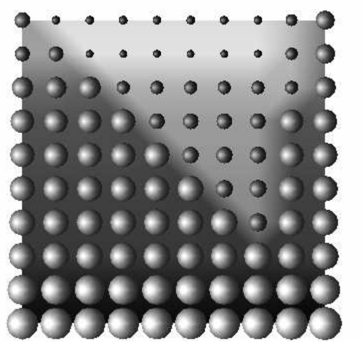
Figure 5. The locations of spheres, representing processors, are determined by the percentages of time spent in each of the three states, busy, idle, and overhead.

The second new visualization was developed as part of an ongoing research project to develop visualizations for new parallel languages that incorporate data distribution semantics, such as High Performance Fortran (HPF) [8] and the parallel C++ language, pC++ [1]. The goal of these visualizations is to provide performance information for programmers specifying data distributions. In particular, the visualizations intend to (1) reveal patterns in how programs access data structures, and (2) give information about whether the distribution of data is effective for that program.

To accomplish these goals, it is necessary to represent both the data structure and processor structure. (In HPF, programmers declare sets of virtual processors which are the targets of data distributions. (See [3] and [8] for more detailed information.) The idea is to track the number of remote and local reads and writes with respect to the data structure and the (virtual) processor structure. Furthermore, it is desirable to have access to both the average and accumulation of those quantities up to a particular time step. These quantities are then mapped onto graphical representations of the data and processor structures in a variety of ways.

Figure 6 illustrates a display in which the cumulative number of remote reads and writes are mapped to the size and color of the glyphs while the average number of remote reads and writes are mapped to the color of the solid sheet behind the glyphs that shows "hot" and "cool" spots in the respective structure. Figure 6 represents a two-dimensional array data structure. Accompanying this display would be a similar visualization portraying the virtual processor topology as maintained by the parallel language. These displays are presented in more detail in [3].

When moving to three dimensional structures, the colored backdrop representing the average becomes more difficult to realize because the third dimension is already being used by the rest of the structure. Thus, in many cases only a single quantity (*i.e.*, cumulative or average) was represented, as in Figure 7. (Note that a glyph's size and color could be controlled by different quantities if so desired.) Also, the volume of the structure in Figure 7 could be rendered, creating a "cloud" which interpolates the data values at the nodes. This could potentially offer insight into the scalability characteristics of the program under examination.



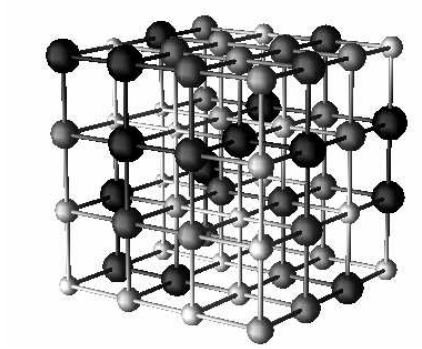


Figure 6. (left) This data distribution visualization represents distributed data elements as spheres whose size and color are controlled by the number and type of references made to the particular element.

Figure 7. (right) Two-dimensional data distribution displays are easily extended to three dimensions, but with some loss of representational capabilities.

Working from Visualization Technique to Visualization Concept. After using Data Explorer for a while, we were impressed by its flexibility, yet we knew that we had only touched the surface of its graphical features. We gradually began to reverse the visualization development process and started using Data Explorer not only as a tool to display a preconceived visualization concept, but as an aid in generating that concept in the first place. This section will offer an example of such a visualization.

Data Explorer offers a realization technique called an *isosurface*. Isosurfaces manifest themselves as contour lines for two-dimensional data (like elevation lines on a topographic map) and as contour surfaces for three-dimensional data. An isosurface represents a set of (actual and interpolated) data points of constant value (an *isovalue*). Such a technique provides an interesting view of larger quantities of performance data that are represented in a three-dimensional, volumetric structure such as a cube. Using this visualization technique as a starting point, we created the display seen in Figure 8. It consists of a set of 64 processors arranged in a 4x4x4 cube. At each processor location, an arrow points in the direction determined by the three-dimensional performance metric [busy, idle, overhead] mentioned earlier. From these three quantities, the "busy" component was used to color the arrows and create the isosurfaces. Each frame of Figure 8 contains a set of isosurfaces for five isovalues distributed between 0 and 1 (the range of values covered by the "busy" metric). The surfaces are colored by the same scale as the arrows. These surfaces give the viewer an interesting perspective on the performance of the parallel application. In addition to revealing the global level of performance within the structure and areas of similar performance levels, the visualization potentially offers insight to the scalability characteristics of the application. (For more information on the use of this visualization technique in performance visualization, see [5].)

In this section, sample visualizations falling into three categories have been presented. First, a scientific visualization package was used to implement well-known visualizations like Kiviat diagrams. Next, two-dimensional displays were enhanced, typically by adding a third dimension to the display. Finally, we created new types of performance displays in this environment. Clearly, performance visualization developers gain access to significant power and flexibility when using existing scientific visualization software.

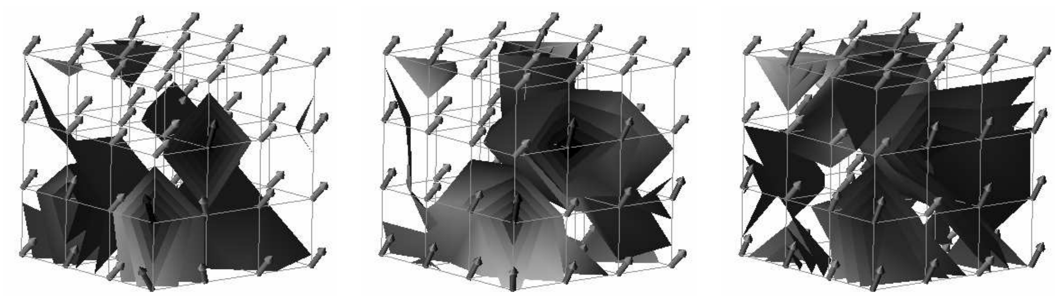


Figure 8. This sequence of images uses isosurfaces to explore the relationships between 64 processors arranged in a cube. This visualization was motivated by the availability a pre-programmed realization technique within the scientific visualization software.

6 Discussion

Having documented several examples, we now turn to a discussion of the possible benefits and detriments of the proposed design process. In addition to the most obvious benefit of significantly reducing development overheads, this technique 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 a majority of their time to building the visualization tool rather than testing visualization usability [14,21]. 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 [7]. 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.

Scientific visualization packages by nature offer a high degree of user control. Thus, the ability to customize displays is built into the package. Other features like display interaction, animation, and modularity are also present. In particular, the ability to interact with what is shown on the screen will become an absolutely necessary component of next-generation visualization tools as displays become more complex and take advantage of multiple dimensions. In summary, these packages have many of the more difficult-to-program features already established and allow visualization developers to focus on the quality of their displays rather than the code needed to generate them.

On the other hand, it is conceivable that generalized data visualization packages contain too much functionality, resulting in “environment overkill.” In other words, the software may be too general, resulting in slower 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; all of our renderings were software generated. Tools built specifically for performance visualization might not have such stressing requirements on the computing system.

7 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 to the arena of performance visualization. Furthermore, the work being done here at the University of Oregon 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.

The work presented in this paper can be and has been extended in several ways. In terms of the high-level methodology we developed, techniques that enable the specification of performance and view abstractions and their subsequent instantiations to performance and view objects are needed. In terms of the visualizations themselves, we are interested in examining how other traditionally scientific visualization techniques may assist in the development of scalable performance visualizations [5]. This would include additional work with graphical representations like isosurfaces and contour lines, for example. Finally, additional work is required to refine a methodology for evaluating visualizations in this environment.

8 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. These results (including color images) and other research work at the University of Oregon can be explored online via the World Wide Web at http://www.cs.uoregon.edu/~hacks/info-research.html.

9 References

[1] F. Bodin, P. Beckman, D. Gannon, S. Narayana, S. Yang. *Distributed pC++: Basic Ideas for an Object Parallel Language*. U. of Rennes.

[2] A. Couch. *Categories and Context in Scalable Execution Visualization*. Journal of Parallel and Distributed Computing, 18, 1, June, 1993, pp. 195-204.

[3] S. Hackstadt and A. Malony. *Data Distribution Visualization for Performance Evaluation*. Department of Computer and Information Science, U. of Oregon, Technical Report CIS-TR-93-21, October, 1993.

[4] S. Hackstadt and A. Malony. *Next-Generation Parallel Performance Visualization: A Prototyping Environment for Visualization Development*, Department of Computer and Information Science, U. of Oregon, Technical Report CIS-TR-93-23, October, 1993.

[5] S. Hackstadt, A. Malony, and B. Mohr. *Scalable Performance Visualizations for Data-Parallel Programs*. to appear Proc. Scalable High Performance Computing Conference (SHPCC) 1994, Knoxville, Tennessee, May, 1994.

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

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

[8] High Perf. Fortran Forum. *High Performance Fortran Language Spec.*, Ver. 1.0. Rice U., May, 1993.

[9] Intl Business Machines Corp. *IBM Visualization Data Explorer, User's Guide*, 2nd ed. August, 1992.

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

[11] 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.

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

[13] A. Malony, D. Hammerslag, and D. Jablonowski. *Traceview: A Trace Visualization Tool*. IEEE Software, September, 1991, pp. 29-38

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

[15] B. Miller. *What to Draw? When to Draw? An Essay on Parallel Program Visualization*. Journal of Parallel and Distributed Computing, 18, 1, June, 1993, pp. 265-269.

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

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

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

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

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

[21] S. Sarukkai and D. Gannon. *Performance Visualization of Parallel Programs Using SIEVE.1*. Proc. of the 1992 ACM Intl Conference on Supercomputing, Washington, D.C., July, 1992, pp. 157-166.

[22] D. Socha, M. Bailey, and D. Notkin. *Voyeur: Graphical Views of Parallel Programs*. SIGPLAN Notices 24, 1, January, 1989. Also, Proc. of the Workshop on Parallel and Distributed Debugging, Madison, WI, May, 1988, pp. 206-215.

[23] 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

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