Visual analysis of stream data

Michael Cheng, Miron Livny, and Raghu Ramakrishnan
Computer Sciences Department
University of Wisconsin–Madison, Madison WI 53706
{mcheng,livny,raghu}@cs.wisc.edu

ABSTRACT

We present the `DEVise`* toolkit designed for visual exploration of stream data. Data of this type are collected continuously from sources such as remote sensors, program traces, and the stock market. A typical application involves looking for correlations, which may not be precisely defined, by experimenting with graphical representations. This includes selectively comparing data from multiple sources, selective viewing by zooming and scrolling at various resolutions, and querying the underlying data from the graphics.

`DEVise` is designed to provide greater support than packages such as AVS or Khoros for this type of application. First, by abandoning the network flow model of AVS and Khoros in favor of a database query model, we are able to incorporate many performance improvements for visualizing large amounts of data. To our knowledge, this is the first attempt to eliminate data size limitations in a visualization package.

Second, by structuring the stand-alone graphics module of most existing tools into user accessible components, users can quickly create, destroy, or interconnect the components to generate new visualizations. This flexibility greatly increases the ease with which users can browse their data. Finally, through limited programming, users can query the underlying data through the graphical representation for more information about the records used to generate the graphical representation.

1 INTRODUCTION

Many applications generate data continuously in the form of a stream. Examples include data from remote sensors, program traces, simulation output, and the stock market. Making sense of data textually is virtually an impossible task. A visual representation, on the other hand, can often be used to look for interesting 'patterns' that are not fully understood *a-priori*. The technique of browsing graphically in order to understand data falls under the domain of *exploratory data analysis*.

In order to exploit the full power of graphical presentation, a visualization environment must address many issues that directly impact its usefulness.

- First, the visualization interface should give the user control over *how* data is to be presented graphically, *what* portions of the data is to be presented, and *where* the graphics should be placed on the computer display. The extent to which the user has control over these three aspects of visualization determines the

*A Data Exploration via Visualization Environment

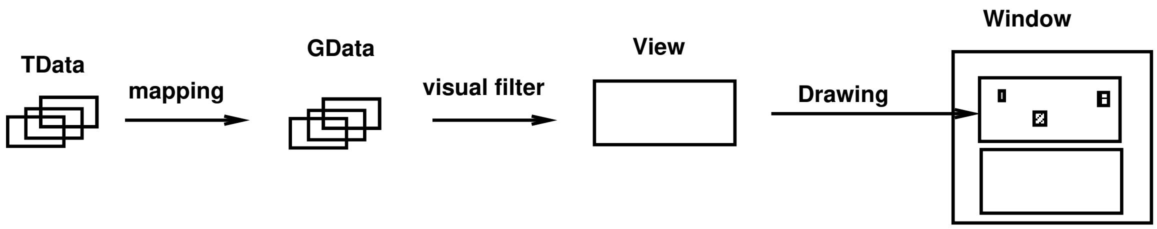

Figure 1: Model of Visualization

flexibility of the visualization software. In addition, the mechanisms for exercising such control through the visualization interface determine the ease of use of the visualization environment.

- Second, the visualization environment should be prepared to handle data sizes larger than available main memory. Many real-life streams are already much larger than available main memory. Even if they can be filtered using some criteria, comparing many streams side-by-side can again overwhelm the virtual memory system of a workstation. The ability to handle large amounts of data enables the user to look at the overall picture, and graphically select (e.g., zoom or scroll) portions of the data for closer examination.
- Third, when looking at the graphical presentation, a user often wants to see the actual data represented by the graphics. A visualization interface should support this functionality.
- Finally, however useful a visualization environment is, it can not be all things to all people. A new type of file not handled by existing code may need to be incorporated, or a new way to draw the graphics needs to be tested. A visualization environment is extensible if it can deal with new situations as they arise.

In this paper, we present `DEVise`, a toolkit aimed at addressing the issues in visualization just discussed. `DEVise` started out as a project to visualize the output from the *DENet*[16] simulation environment. It gradually evolved into the general purpose visualization toolkit that it now is. We note that this is the first visualization toolkit that we know of that is designed to eliminate data size limitations.

In Section 2 we present a model of visualization designed to give the user finer control over the process of visualization compared to using a stand-alone plotting program. In Section 3 we give a tour of `DEVise` using two motivating examples that are designed to illustrate its flexibility and its ability to handle large amounts of data. In section 4 we will describe in further detail the functional components of `DEVise`. We will also address ease of use issues through a discussion on the design of the control panel. In Section 5, we describe how `DEVise` supports extensibility, including querying of the underlying data, adding a new type of file, and creating a new method of displaying data. Finally, we conclude with related work and a discussion of future work.

2 MODEL OF VISUALIZATION

Instead of providing a stand-alone graphics package, we structure visualization into a four-stage process, as shown in Figure 1. As we shall see in Section 3, this division gives the user flexibility in controlling each step of the visualization process.

TData is the textual data input to our model as provided by the user. It consists of attributes stored in records. A *stream* is a TData object that allows the continual insertion of additional records even while it is being visualized.

The user specifies how TData is to be mapped to its graphical representation, the *GData*, through a *mapping*.

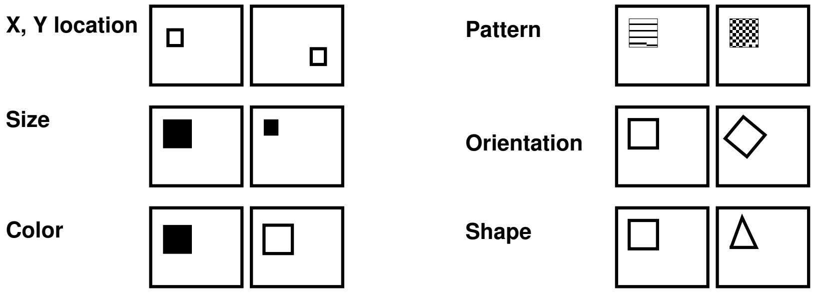

Figure 2: Graphical Data Attributes

Each TData record is mapped to a GData record, where each GData record consists of graphical attributes *x, y, size, color, pattern, orientation,* and *shape*, as shown in figure 2. This definition of a *mapping* gives the user control over the attributes of graphical representation down to the record level. This is considerably more flexible than just specifying a single color or shape to be used for all records, as is the norm in plotting packages.

A *view* is used to display the GData, and its accompanying *visual filter* determines the portion of GData to be displayed. The visual filter is defined as a query over the graphical attributes of GData. Scrolling or zooming are accomplished by defining a range query over the x and y attributes of the visual filter. However, a *visual filter* can in practice be more powerful than just defining a scroll or zoom. It can be an arbitrary query over all attributes of GData, such as "Retrieve all GData where color = "red" and shape = "rectangle", and ...".

The view takes a collection of GData records as defined by the visual filter, and draws them in a *window*, the physical output device. Different *views* implement different ways to draw the GData. For example, a view can eliminate GData records that are mapped to the same location in a window. Or it can keep a count of the number of GData records that fall in the same range, and display a 2D histogram. More than one GData can also be displayed overlapped in the same view.

A window provides the real screen estate for views to draw GData. If more than one view is inserted into a window, it is also responsible for arranging them inside its boundaries. For example, a window can arrange the views vertically, or horizontally.

3 A TOUR OF DEVise

In this section we introduce the functionality of `DEVise` via two motivating examples. First we describe how `DEVise` is used as a debugging tool for distributed scheduling algorithms in a materials science application, and then as a browsing tool for monitoring file server activities. The generality of the model presented in section 2 enables us to apply `DEVise` for such different applications. Furthermore, the extensibility of `DEVise` helps us address the specific needs of these two application.

For the material science application, we emphasize the dynamic mapping capability of `DEVise`, which enables us to experiment with different representations of the algorithms. For the file browser application, we emphasize the ability of `DEVise` to handle large amounts of data.

3.1 Material Science Distributed Scheduling

We have an application for a material scientist that involves the calculation of energy levels at 30 distinct points over a certain material. The calculations are iterative. At each iteration, the energy level at each of the 30 points is calculated independently, where each calculations can take anywhere from $\frac{1}{2}$ to 7 minutes, after which the results are gathered before starting the next iteration. The whole computation takes approximately 6 hours to complete.

For performance reasons, the application obtains idle workstations on which to schedule the independent calculations. The order in which the calculations are dispatched to the workstations can significantly impact the performace of the algorithm. Here we use `DEVise` to compare two different algorithms. The first algorithm maintains statistics about the time spent by each of the 30 calculations, and schedules those that take the longest first. The second algorithm dispathes the calculations in random order.

The flexibility and extensibility of `DEVise` give us the ability to fine tune the visualizations in ways not available in existing packages. The ability to define mappings at the record level allows us to experiment with different color and shape schemes until we arrived at different views that we think best represent some aspects of the computations. The extensibility of `DEVise` gives us the ability to query the data records that underly the graphics.

Photo 1 is a snapshot of the visualization comparing these two algorithms. We have plotted on the X axis time expressed in seconds, and on the Y axis the worksation number. There are two types of plots for each algorithm. The first plot draws a rectangle for each calculation performed on a workstation. The width of the rectangle represents the amount of time taken by the calculation. The position of the rectangle on the Y axis indicates the workstation where the calculation takes place. The color of the rectangle represents how it is scheduled. For example, the color black indicates that the calculation has to be aborted because a user has returned to reclaim the workstation. The color green tells us the calculation is the first to be scheduled on the workstation for a particular iteration, while the color blue means it's the second calculation to be scheduled on the same workstation in a particular iteration. In general, the more green the better because scheduling more than one calculation on the same workstation impedes performance. For convenience, we call this first type of plot the *activity* plot. For the second plot, the red areas indicate the times when a workstation is idle, while waiting for others to finish the remainder of the iteration before starting the next one. We call this the *idle plot*. Notice that both plots show that the algorithms have distinct iterations.

It is obvious from the graphics that the first algorithm fares better than the second. And the graphics tell the story in three ways. First, observe that the idle plot for the first algorithm has much less red areas than the second algorithm. This means that the first algorithm is spending more time doing useful computation. Second, observe that the activity plot of the first has less white space in between iterations. Again, this is an indication that more time is spent on useful computation. Finally, observe that the activity plot of the second algorithm has many blue rectangles, indicating that workstations usually have to perform more than one calculation per iteration, while the first algorithm usually schedules only one calculation per workstation in each iteration.

Photo 2 shows another snapshot of the visualization. This time we have zoomed into a trouble spot in the computation for closer examination. The X axis of the views are linked so that zooming one view zooms all other linked views. Clicking on the mouse with the middle button gives us information about an aborted calculation. Notice that there are two records that underly the black rectangle. This querying capability has proven invaluable for debugging the algorithms during their design phase.

3.2 Monitoring File Servers

We have an application in which data are gathered every minute from a group of five file servers. About 4 megabytes of data are gathered per month from each file server. The data capture both the amount of disk transfer, and the time spent for various activities at the file server, including user, system, and idle times.

In a typical use of this data, we wish to monitor the file server activities. This is an application that exercises the ability of `DEVise` to handle data sets larger than memory. With 5 file server, there are 20 megabytes of data to be monitored per month. An application that simply reads all data into virtual memory can quickly overwhelm the virtual memory system of a modest workstation. For example, on a DECstation 5000 with 16 megabytes of real memory, it takes about 11 minutes to display 15 megabytes of data in virtual memory, and only 1 minute to display the data without placing them in virtual memory, an order of magnitude difference.

Photo 3 compares the idle and disk activities between two file servers named "ren" and "tweety". In the window "Devise_Window_0", the X axis represents time, and the Y axis represents either the idle or disk activities. Color is used to represent system activity: whenever system activity is high, the color is red, otherwise it is blue. The plots show that both "ren" and "tweety" go through similar cycles of low idle and high disk activities. These cycles occur once a day, late at night, during file backup operations. The user can discover the times of these cycles by clicking on the middle mouse button, which produces a pop-up window that displays time in a format such as "Wed Dec 7 22:54:16 CST 1994" instead of an integer that conveys no meaning.

Notice that "ren" has many periods of high system activity, as represented by a large number of red rectangles. To better understand the effect of high system activities, we arranged scatter plots of disk versus idle in the window "Devise_Window_1", with the color red again representing high system activity. Intuitively, we expect to see a roughly inverse relationship between idle and disk activities: whenever the file server is not idle, it is serving file requests. This correlation is higher for "tweety" than for "ren". And much of the variations in "ren" can be attributed to high system activities.

The plots in Photo 3 bring more questions than answers. For example, how often do high system activities occur, and how high do they go? Are there any other activities correlated with system activities? To answer these questions, we generated Photo 4, showing plots of user, system, idle, and disk activities versus time for the file server "ren". To facilitate navigation, there are three separate links on the views. First, all the X axes are linked. Next, the Y axes of the "disk" plots are linked. Finally, the Y axes of the "system", "user", and "idle" plots are linked.

Observe that for the file server "ren", not only are high system activities closely related to low idle activities, they are also closely related to high levels of user activities. In general, high system and high user do not occur at the same time. Rather, the plots show that high system and high user activities occur very close together in time. We do not yet know what causes the difference in behavior between "ren" and "tweety". But this is one example where visualization helps us see a relationship which we did not expect *a-priori*.

4 FUNCTIONAL COMPONENTS IN DEVise

Our current implementation of `DEVise` is sufficiently complete that most users can start using it without programming. In this section, we will first discuss how the user interacts with `DEVise` to construct their visualization. Then we will describe in detail the components currently available to users. These components include: TData, Mapping, View, Window, Visual Link, and the Control Panel.

Hidden under the visible components is the underlying architecture that supports all of the functionalities of `DEVise`. This includes a buffer manager and a query processor to support access to large amounts of data, a GData component that automatically stores GData converted from TData, and components that access the windowing systems, and intercept user events. The discussion of the underlying architecture is beyond the scope of this paper, and shall be presented in a future manuscript.[5]

4.1 Interacting with DEVise

`DEVise` provides a *structured interface* to help the user generate visualizations. It is designed to take the user from the initial file to the final visualization step-by-step, reducing the amount of information that the user has to know at each step. For both the material science and file server examples in section 3, the user is first asked to select an input file by clicking on the file name. The user is then asked to select a file type for the input file. This step imports the file into the application as a TData object. Next, the user is asked to select a mapping to map TData records into GData records. This can be a predefined mapping, or one that the user creates dynamically. (See section 4.3) After creating the mapping, the user is asked to pick a view to display the GData, to specify the initial values for the visual filter, and to pick a window to place the view. Finally, the user has the option to link views together.

After creating the visualization, the user has the flexibility to manipulate it even further. Additional files can be selected for visualization. Additional mappings for TData can be created, or an existing mapping modified to experiment with graphical representations. A view can be removed from its window to reduce clutter, or be moved to a different window so that different views can be brought together for side-by-side comparison. A view can be removed from the group of views with linked axes, or be linked with a different group of views. Scrolling and zooming are controlled by buttons on the control panel, by pressing on keys with the mouse pointers inside a view, or by pointing and dragging the view with the left mouse button to select a rectangle to zoom in on. It is this exploratory nature of investigation that helps us to identify the potential problem with high system times for the file server example.

Whenever necessary, we can customize `DEVise` to meet application specific needs. By default, clicking in the window with the middle mouse button displays the current x and y coordinates. For the material science example, we had to override the default message with one about the data records that underly the mouse button. For the file server example, we had to display the date in the required format. As we shall see in Section 5, the programmer does not have to intercept the mouse events. He or she only has to generate the message when asked to do so by the toolkit.

In Section 1, we made the point that a visualization environment should allow the user control over *what* is to be visualized, *how* the data is visualized, and *where* the data is to be placed on the display. This functionality is achieved in `DEVise` as follows: *what:* by selecting the TData, setting the visual filter of views, and linking views; *how:* by experimenting with mappings; *where:* by placement of views in windows. We believe that the run-time flexibility offered by `DEVise` greatly improves the ease with which the user can concentrate on generating visualizations that lead to a deeper understanding of the data.

4.2 TData

The current implementation of `DEVise` allows the user to import ascii files of varying formats without programming. The user first creates a schema file that contains the specification of the file format. The file format is then imported into `DEVise` as a file type, which can be used to read all input files with that format.

The schema file can specify the file format for both ascii and binary files. Currently we only have ascii TData implemented. We have also incorporated several techniques to speed up access to large ascii files. First, an index to the beginning of each line in the file is built on the fly to reduce the time for random access to records in the file. Second, the index is automatically written to disk when the DEVise exits, and read automatically when DEVise restarts to reduce the time to start a program. And finally, if the file has been appended, the new index is an extension of the old index, i.e, not rebuilt from scratch.

4.3 Mapping

The default mapping provided by DEVise is the interpreted mapping. The user specifies the value to be taken by each of the GData attributes by typing an expression in the tcl[22] language. The expression can make use of TData attribute names, which are converted into tcl variables automatically by the interpreted mapping. The interpreted mapping is designed to give users the ability to dynamically explore graphical representations. If the user is not satisfied with the current mapping, he or she can always re-edit the mapping and the graphics will be updated automatically using the new mapping.

For the file server example in section 3.2, the user uses the following mapping to plots idle versus time:

```
x: $time
y: $idle
color: [ if { $system > 50 } { set color 2 } else { set color 3 } ]
```

In addition to the interpreted mapping, DEVise also supports compiled mappings. Users concerned about performance can compile and link C++ code segments with the DEVise library. However, this step shall be eliminated in the future, when DEVise supports automatic compilation of the interpreted mapping.

4.4 View and Window

DEVise currently implements two different types of views. The first is called a *Scatter* view, which draws all shapes without implementing any optimization. The second is called *Sorted X* view. This view assumes that a large amount of data is to be drawn, and the data is sorted on the X attribute. As a result, those GData records mapped to the same location on the computer screen are eliminated so that only one GData is drawn.

Currently there is only one type of window available in DEVise. It stacks all the views vertically within the window boundaries. The user can dynamically remove any view from within a window, or move a view to any other window.

4.5 Visual Link

Recall the motivating examples in section 3, where the views are linked so that zooming or scrolling one synchronously zooms or scrolls another. Visual linking in DEVise goes beyond just the linking of the x or y attributes of GData: any subset of the x, y, color, size, shape, orienataion, and pattern attributes can also participate in visual linking.

To illustrate the usefulness of linking by attributes other than x or y, we go back to the filer server example. Suppose the user wishes to compare the idle activity for the same server at two different time periods. Naturally, the X axes of these two views are not linked. However, these two views can still be linked by color. The user can specify to see only "those shapes in red" in the first view. And this command affecting the color attribute of the visual filter of the first view automatically affects the visual filter of the second view.

4.6 Control panel

The *control panel* implements the mechanism through which the user exercises control over the components used in visualization. In our attempt to make the visualization environment flexible, we have also introduced many options for the user when generating a visualization. To minimize the amount of information for the beginning user, and to make the flexibility available to experienced users, we have adopted three strategies. First, the structured interface, as discussed in section 4.1, takes the user step by step to generate a visualization. Second, the use of menus hides options that users will not normally use. For example, the view menu allows users to remove views from windows, but only when it is asked for. And finally, the use of start-up file allows the user to specify default parameters for views and windows.

Our design and implementation of the control panel is far from complete. The reason is that the design of the user interface is an iterative process. The control panel is continually being updated as we get from end users feedback about how they want to interact with it. For example, we still have no option for the user to say, "plot this file using all default parameters".

5 EXTENSIBILITY

`DEVise` is structured as a hierarchy of C++ classes. We have developed a set of well-defined interfaces for each component of the basic visualization model, and for the additional components that we have added to improve either performace or functionality. These components are: *TData, GData, mapping, buffer manager, query processor, view, window,* and *visual links* for views.

The extensibility of `DEVise` comes from the ability to create a subclass of an existing class, and then register the new (sub)class with `DEVise`. For example, to read a new type of file, we can simply implement a new subclass of TData and register it with `DEVise`. Once registered, all the existing components of `DEVise`, such as mappings, view, and windows, can be used in the visualization of this new type of data. The same approach is used to add any new component. One added advantage of the subclassing approach is that it is useful for generating a prototype. Whenever a new component is needed, a quick and inefficient implementation can be tested before undertaking a full implementation. We used this technique throughout the implementation of `DEVise`.

Querying the TData that underlies the graphical presentation involves three steps: intercepting the events that are interpreted as a query from the user, finding the TData for the query, and presenting the result to the user. Currently `DEVise` provides support only for the first and third step. For the first step, whenever the user clicks a button or draws a rectangle inside a view, the events are translated into a call to a class instance responsible for acting on these events. For the third step, `DEVise` is able to create a pop-up window to display information specified by the program. By default, clicking the middle mouse button creates a pop-up window to display the current X and Y locations in the view, and sketching a rectangle in the view zooms in to the sketched region. The programmer can override the actions of the default class by defining a new subclass to perform his or her own actions.

Currently `DEVise` has no support for finding TData that underlies a given region in a view. This is a weak link in `DEVise` because we depend on the programmer to find the inverse mapping from graphics to TData, using either the buffer manager interface or the TData interface. The amount of programming can vary from trivial to extensive, depending on the complexity of TData and the mapping. Work is in progress to update the query processor to perform this type of query.

6 RELATED WORK

An introduction to existing visualization software can be found in the survey by Kornbluh.[15] Perhaps the most widely used visualization systems today are AVS[6] and Khoros.[14] They are quite successful at visualizing data from physical phenomena such as weather patterns, terrain maps, or medical images. However, they have the limitation that when the number of records grows large, the use of virtual memory results in excessive paging. Flexibility in experimenting with the graphics and arranging the display is also quite limited.

There is a large body of literature on techniques used for exploratory data analysis and analysis of multi-dimensional data.[1,3,4,11,17,20,25,26,8,27] For specific techniques used to visualize trace files, we refer the reader to GMAT,[19] Traceview[18] and Paragraph,[12] and Pablo.[7] Common to exploratory data analysis (as used by statisticians) and trace file visualization (as used by computer scientists) is the need for a powerful browser. Our contribution is in making the data browsing aspects of visualization flexible, and in making a first attempt at visualizing large data sets without resorting to virtual memory. We refer to a computational fluid dynamics example[10] as another application that also encounters problems with virtual memory.

The flexibility we offer users comes partly from the ability to define dynamically mappings from data to graphics attributes. Our choice of graphical attributes is taken from Bertin's work on graphics information processing.[2] Bertin's static attributes are even more uesful when incorporated as dynamic attributes in our system. For work on mapping from a general data structure consisting of arrays and structures into graphics attributes, we refer the reader to the VIS-5D and VIS-AD systems.[13] `DEVise` differs from these systems by concentrating on visualizations of the simpler stream data structure. This allows `DEVise` to provide a more flexible mapping mechanism, and to incorporate performance optimization specific to strem data.

The trade-off between a tightly coupled system and a loosely coupled system is discussed in the paper about the Sprite IIIDPC[21] system. The tightly coupled Sprite IIDPC system "helped increase the amount of data and the speed which it could be displayed", while its companion, the Sprite III[9] system "allows users to reorganize and transform their data into any number of configurations". This is also our philosophy when designing `DEVise`. The tightlty coupled design of `DEVise` allows us to incorporate many performance improvements that would otherwise not be possible in a loosely coupled system.

7 CONCLUSION

In this paper we have presented `DEVise` as a tool that can be used for browsing large amounts of stream data. The flexibility offered by `DEVise` is that the user gets complete control over *what* is to be visualized, *how* data is to be visualized, and *where* the visualization is presented on the computer screen. In addition, `DEVise` incorporates many techniques that are designed specifically to allow users to visualize data sets larger than memory. Finally, `DEVise` is extensible. By using the toolkit, programmers can define new types of data files, new ways to display the graphics, and add queries to the underlying data.

Although functional, `DEVise` can still be improved in many significant ways. First of all, the control panel needs to be upgraded to supply a good set of defaults so that the beginning user can generate a simple visualization immediately. The ability to use the more complex features of `DEVise` can then be acquired over time as the user learns its capabilities.

Second, we need to develop the notion of a *visual query* by decoupling the parameters used in visualization from the underlying data files. This would allow the user to experiment with the visualization using a sample file, and then apply the same visualization to all files of interest.

Finally, although the framework of `DEVise` has been established, substantial components have to be added to make it into a complete visualization environment, as opposed to a toolkit. As discussed earlier, `DEVise` needs to be updated to provide even better support for querying the underlying data from the graphics. It can also be extended to make the interpreted mapping run faster, and to support a greater variety of built-in shapes, and views.

Currently `DEVise` is being incorporated as a front-end for the `SEQ` sequence querying project[23,24] at the University of Wisconsin. `SEQ` and `DEVise` together form a powerful visualization and data analysis environment with `SEQ` as the data analysis back-end, and `DEVise` as the visualization front-end. As such, we plan to use them to extract remote sensor data for finding suitable parameters in weather model experiments, and for analysis of stock market data.

8 ACKNOWLEDGEMENTS

Special thanks to Jim Pruyne for the material science example, and to Mike Litzkow for the file server example. Michael Cheng's work was supported in part by an IBM Fellowship, and NASA grant NAGW-3921. Miron Livny's work was supported by NASA grant NAGW-3921. Raghu Ramakrishnan's work was supported by a Packard Foundation Fellowship in Science and Engineering, a PYI Award with matching grants from DEC, HP, IBM, Tandem, and Xerox, NSF grant IRI-9011563 and NASA grant NAGW-3921.

9 REFERENCES

[1] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29:127–142, May 1987.

[2] Jacques Bertin. *Graphics and Graphic Information-Processing*. Walter de Gruyter, Berlin, 1981.

[3] Leo Breiman and Jerome Friedman. Tools for large data set analysis. In Edward J. Wegman and James G. Smith, editors, *Statistical Signal Processing*, pages 191–197. Marcel Dekker, Inc, 1984.

[4] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large n. *JASA*, 82:424–436, 1987.

[5] Michael Cheng, Miron Livny, and Raghu Ramakrishnan. Visual querying in devise. manuscript in preparation.

[6] David Kamins et al. Craig Upson, Thomas Faulhaber Jr. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics & Application*, 9(4):30–42, July 1989.

[7] Ruth A. Aydt Daniel A. Reed and Tara M. Madhyastha et. al. *An Overview of the Pablo Performance Analysis Environment*. Computer Sciences Department, University of Illinois-Urbana Champagne, 1992.

[8] S.H.C. du Toit, A.G.W. Steyn, , and R. H. Stumpf. *Graphical Exploratory Data Analysis*. Springer-Verlag, New York, 1986.

[9] David P. Edwards, David Bridenstine, and Laurie Littledike. Configurable data analysis and visualization system. In *Visual DataExploration and Analysis, Proceedings SPIE*, volume 2178, pages 48–57, February 1994.

[10] Al Globus. Optimizing particle tracing in unsteady vector fields. In *Visual DataExploration and Analysis, Proceedings SPIE*, volume 2178, pages 127–133, February 1994.

[11] E. James Harner. An exploratory data analysis and modeling environment based on lisp-stat. In *Proceedings of the Statistical Computing Section*, pages 70–78, 1991.

[12] Michael T. Heath and Jennifer A. Etheridge. Visualization the performance of parallel programs. *IEEE Software*, pages 29–39, September 1991.

[13] William Hibbard, , Brian E. Paul, David Santek, Charles Dyer, Andre Battaiola, and Marie-Francoise Voidrot-Martinez. Interactive visualization of earth and space science computation. *IEEE Computer*, pages 65–72, 1994.

[14] The Khoros Group, University of New Mexico, Albuquerque, NM 87131. *Khoros Manual*, 1991.

[15] Ken Kornbluh. Active data analysis: advanced software for the 90's. *IEEE Spectrum*, 31(11):57–83, November 1994.

[16] Miron Livny. Delab- a simulation laboratory. In *Proceedings of the 1987 Winter Simulation Conference*, Alanta, GA, December 1987.

[17] David J. Lubinsky and Forrest W. Young. Guiding data analysis. In *Proceedings of the Statistical Computing Section*, pages 205–210, 1990.

[18] Allen D. Malony, David H. Hammerslag, and David J. Jablonowski. Traceview: A trace visualization tool. *IEEE Software*, pages 19–28, September 1991.

[19] Mary E. Zosel Mark K. Seager, Nancy E. Werner. Graphical analysis of multi- and microtasking execution on cray multiprocessors. In Gregory M. Nielson and Bruce Shriver, editors, *Visualization in Scientific Computing*, pages 160–173. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[20] John Alan McDonald, Werner Stuetzle, and Andreas Buja. Painting multiple views of complex objects. In *ECOOP/OOPSLA '90 Proceedings*, pages 245–257, October 1990.

[21] Shaun M. Oborn and Dean S. Garlick. Effective visualization tools for large data sets. In *Visual DataExploration and Analysis, Proceedings SPIE*, volume 2178, pages 40–47, February 1994.

[22] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, Massachusetts, 1994.

[23] Raghu Ramakrishnan, Michael Cheng, Miron Livny, and Praveen Seshadri. What's next? sequence queries. In *Proceedings of the International Conference on the Management of Data (COMAD)*, December 1994.

[24] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. Sequence query processing. In *Proceedings of the ACM SIGMOD '94 International Conference on Management of Data*, May 1994.

[25] Werner Stuetzle. Plot windows. *JASA*, 82(398):466–475, 1987.

[26] Luke Tierney. *LISP-STAT: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics.* John Wiley and Sons, New York, 1990.

[27] Paul F. Velleman. Computing and modern (exploratory) data analysis. In *Proceedings of the Statistical Computing Section*, pages 46–54, 1990.