We Need Interactive Data Interpretation Rather Than Interactive Data Visualization

A Position Paper for the Workshop on Scientific Visualization Environments at Visualization '91

Gudrun J. Klinker

Cambridge Research Lab,
Digital Equipment Corporation

August 30, 1991

1 Introduction

General-purpose visualization systems, such as AVS [7], have recently received a lot of attention in the visualization community. However, many application programmers still prefer writing their own visualization code over using a standard visualization system. Why? What features are missing from current systems that prevent real applications from using them? In this position paper, we emphasize that certain application areas, such as robotics or medical imaging, need an interactive data interpretation component in addition to straightforward data visualization. We use the term "interactive data interpretation" to describe the process of generating models from data measurements which are often provided in multi-dimensional, multi-modal arrays. The process is sometimes also referred to as "inverse modeling". Typical examples of data interpretation would be determining the outline of an object in a two-dimensional image or fitting a surface model to a three-dimensional volume. In either case, the process could be performed completely automatically (the typical case in computer vision) or partially interactively by the user, as done in [1] to outline a particular type of cell in electromicroscope data.

We present a list of basic features that need to be included in today's visualization systems to make them useful for interactive data interpretation. We draw from our experience of having implemented a toolkit, VDI [5], for visualizing computer vision applications. We are currently in the process of transferring the major concepts of VDI into AVS. We expect such extensions to AVS to be useful in many application areas, such as computer vision, medical imaging and seismic data analysis.

2 Merging Geometric Data with Array Data

A key issue in interactive data interpretation is the combination of geometric and array data. As discussed above, data interpretation systems often use large amounts of multi-modal, multi- dimensional array data. The user is interested in visualizing and (manually and/or automatically) interpreting the arrays, potentially by combining information from several data sources and by understanding the relationships between several arrays. The interpretation results are often represented in geometric form and they need to be overlaid onto the array data such that the user can evaluate their quality. Some research on mixing array data with geometric data is currently under way [4]. But so far, standard visualization systems provide little support for merging the data formats.

3 Data Format vs. Data Representation

Traditionally, data format and data representation have been tightly coupled: arrays have been displayed as images or volumes while geometric models have be rendered as lines and surfaces. Systems also often make assumptions that relate presentation styles to the dimensionality of the data and to specialized, pre-defined viewing preferences: image sequences are shown as movie loops, color data is shown using the color capabilities of the display, etc. However, this close connection between data format and representation is overly restrictive. Array data can be

shown both in an intensity- based display mode and as geometric data, e.g.: as graphs, terrain maps (surfaces), flow vector fields, or as printed numbers. Sometimes, a user may be interested in viewing a color image as a movie loop of the three color bands, or three selected black- and-white images of a time sequence as the red, green and blue bands of a single color image. Decisions on how to display certain array data should be made by the user of the visualization environment by selecting appropriate parameter values rather than be hardwired into the system. Below follows a list of what we consider to be essential features of a future data visualization and interpretation system.

3.1 Variable views of array data

When looking at a particular (high- dimensional) array of data, users may want to see it in various different ways, depending on their current interest. The users should be able to specify the viewing parameters interactively. Such parameters include, of course, the typical cropping, zooming, subsampling and color mapping selections that are available in most current systems. Beyond that, there need to be mechanisms to specify how many (and which) dimensions of a high- dimensional array should be displayed and how the remaining (undisplayed) dimensions are to be treated. Several issues are involved in this data reduction process.

First, we need to specify along which dimensions we want to reduce the data. In the simplest case, the data is reduced exactly along the undisplayed dimensions. However, users may prefer to reduce the data along a linear combination of array dimensions. Such mechanisms can be found in today's volume visualization systems, displaying arbitrarily oriented 2d- slices of 3d-volumes. The mechanisms will have to be extended towards higher- dimensional arrays.

Second, the data reduction procedure must be specified for each reduced dimension. Should the data be accumulated (projected) along a particular reduction direction, or should it be sliced, or should some or all of the data values be kept in a vector and displayed simultaneously using some encoding scheme or sequentially as a movie loop? If the data is to be sliced, the slice position needs to be defined. In the case of data projection, the projection function needs to be specified. Today's visualization systems typically use additive accumulation along the projection direction (e.g.: to display partially transmittant volumes). However, a statistician working with arrays of probability data might prefer multiplicative accumulation. And even other accumulation functions are conceivable. Multiplying the data elements by weights that are a function of the position and value of each element is also useful. Furthermore, users may want to specify their own functions to compute interpolated or subsampled values along the projection direction from the grid of data elements.

Finally, the question arises how multi- variate information is to be presented on the screen. As mentioned in the beginning of this section, the data could be presented in many different ways, such as in an intensity- based display, as a graph, or as printed numbers. For intensity- based displays of arrays with more than one element per data position, the data could, for example, be shown in a movie loop, as a color image, a stereo image (viewable through special glasses), a blink comparator alternating between two (or more) values per pixel, or by iconic symbols such as flow vectors or stick symbols[2], potentially combined with acoustic output. If the data is shown as a graph, several values per pixel can be overlaid as several graphs in a single display. For printed numbers, the data vector can be printed out at every displayable data position. Current systems typically make assumptions as to what kind of data is to be displayed in what dis-

play style. We suggest to leave such decisions up to the user. The user interface should provide mechanisms for the user to specify by what display mechanism the information is to be presented. Furthermore, the interface should provide the means for specifying how the data is mapped onto any particular data representation. For example, a user might want to use a logarithmic mapping function for relating data values to intensities or to the shape of iconic symbols.

3.2 Multiple views of geometric and array data

For many reasons, users may be interested in analyzingrelationships between data from one or more data sources that are displayed simultaneously in one or more windows: on the one hand, they may want to look at a single array, displayed in several different representations, as discussed above, to benefit simultaneously from several different visualization cues. On the other hand, they may be interested in investigating relationships between several arrays and potentially some geometric data that may be the result of previous data interpretation steps. In that case, it might sometimes be most convenient to overlay all data in a single display window, whereas at other times, a side- by- side presentation in several windows is more helpful. In most general terms, users may want to visualize *n* sources of information in *m* windows, some overlaid and some shown side- by- side. An interface for mapping arrays to windows and for specifying relationships between windows will be needed.

If a relationship between several display windows is established, the visualization system has to provide aids for the user to relate a pixel position in one window to positions in the other windows. Such aid can be provided via cursor- linking. For presenting relationships between several data sources in a single window, we suggest two mechanisms: Users can either overlay the selected arrays in the window, or they may want to use some of the arrays as masking operations on the other arrays. In the latter case, the masks need not be displayed but they rather mask out some of the data in the remaining arrays. In all cases, we need to consider how to show relationships between arrays with different offsets, sizes and dimensions. As before, we suggest to leave the decisions up to the user and to provide an interface via which the user can specify linking information between combinations of arrays. This requires an interface to specify coordinate transformations between arrays. If arrays of different dimensionality exist, users have to define whether they want to reduce the data of the higher- dimensional array, or whether they prefer spreadint the information of the lower- dimensional array over more dimensions.

Besides considering variable array formats, we also need to expect that different data sources will be shown in different graphical representations. How about overlaying a terrainmap, printed numbers or a vector flow field from one array on an intensity display of another array? Similarly, users may want to overlay geometric data, such as the rendering of a scalpel or a medical probing device, on intensity data (e.g.: X- ray data). The data sources now have to be zoomed individually such that all renderings can be shown in appropriate size.

4 System Architecture

A visualization environment should primarily be a very generalframework which users/programmers can customize as needed. The framework mainly provides a few, key capabilities and modules plus the infrastructure for transferring any kind of data between any processes. The infrastructure and the modules are used to visualize and interpret any kind of data in its relationship to any other data in the system. The system needs to provide several implementation layers such that an application programmer can freely customize the environment while the final user of the system sees a "turn- key" system adapted to the environment in which it is operating. We expect that the commonly used visual programming paradigm (e.g.: in AVS) may serve as a useful framework, if it is extended appropriately. Visual programming encourages system modularity. Such modularity allows programmers to customize the system, replacing some particular modules by their own experimental code while still benefiting from the capabilities of the rest of the system.

4.1 Module hierarchies and automatic network customization

Generating large networks from many small modules may become quite cumbersome, and the resulting network may be hard to visualize on a screen. To deal with large networks, the system needs to provide interactive mechanisms for the user to group modules into meta- modules, thus creating module hierarchies. Beyond simply replacing a group of module icons by a single new icon, such a mechanism may incorporate capabilities for recompiling the code of the initial modules into a single new code segment, thus improving performance by reducing the communication cost between the initial modules [3].

Furthermore, the user must be allowed to set the networks up automatically. AVS provides mechanisms to save networks and read them back in, as well as a scripting system via which an entire user interaction with a network (setting up the network plus changing parameters interactively) can be preplanned. Beyond such capabilities, users might require that modules can send preferred parameter defaults along with the data to other modules.

4.2 Mechanisms for incremental data interpretation

If a user wants to use the visualization environment for interactive data interpretation, in addition to mere visualization, the currently typical data flow environment needs to be extended in two important ways:

First, the network needs to allow for circular data flow: the interpretation process generates results which may need to be improved or extended incrementally. For example, a physician may start by roughly outlining a particular cell in an electromicroscope image and then incrementally generate more and more precise outlines by using semi- automatic mechanisms like snakes [1] until the contour exactly matches the cell boundaries. AVS has recently added circular data flow capabilities to its visual programming paradigm. We are currently investigating its usefulness for computer vision.

Second, incremental interpretation requires permanent storage capabilities: users may need to store intermediate results from some or all iterations since they may want to go back to previous stages of the interpretation process (backtracking) and embarque along different, more promis-

ing directions of data interpretation than the direction previously pursued. Already today, users of visualization systems can organize their own permanent storage capabilities, e.g.: by saving relevant iterative data in files. However, to spare the application programmer the burden (and inefficiency) of maintaining such data files, the visualization system should provide access to data bases.

4.3 Customizing the visual programming environment

Apart from user- extensible module libraries and an interactive network editor, other parts of the visual programming environment should be customizable as well. Most importantly, there is the module scheduling engine which decides on the sequence in which runnable modules will be activated. In data flow networks with circular data flow and shared, incrementally changeable data, the sequencing of module activation has direct effect on the interpretation results. Furthermore, users may be interested in running their data flow network on a set of parallel machines. In both cases, the users will need to experiment with the scheduling algorithm.

Second, the screen layout should be customizable. Currently, the display screen gets cluttered with information from many different contexts, such as the data flow network, the visualization display windows, plus windows (such as electronic mail) from the operating system. With multi- headed screens emerging, the user should have the freedom to divide the screen space more wisely.

Finally, the set of widgets that are associated with a particular parameter type should be programmer- extensible. The choice of appropriate widgets can be crucial to the look- and- feel of applications, making a system acceptable or unacceptable to a user group. We doubt that the developers of a visualization environment will be able to foresee all potential uses of widgets for all applications. Just as X provides mechanisms rather than policies [6], visual programming environments should provide a framework to which many different styles of interaction can be attached.

Acknowledgments

Thanks to all permanent and temporary members of the Visualization Group (Ingrid Carlbom, Richard Szeliski, William Hsu, Keith Waters, David Tonnesen, Demitri Terzopoulos, and Stephane Lavallee) who have participated in discussions on how to visualize the data used in our group.

References

1. I. Carlbom, D. Terzopoulos, and K.M. Harris. Reconstructing and Visualizing Models of Neuronal Dendrites. *Proc. CG International'91: Visualization of Physical Phenomena*, Boston, MA, June 24- 28, 1991, Springer- Verlag, Tokyo, Japan.
2. G. Grinstein, R.M. Pickett, M.G. Williams. EXVIS: An Explanatory Visualization Environment. *Graphics Interface*, pp. 254- 261. London, Ontario, Canada, June, 1989.
3. N. Hunt. IDF: A graphical data flow programming language for image processing and computer vision. Technical Report TR- 90- 05, Teleos Research, 576 Middlefield Road, Palo Alto, CA 94301, August 1990.
4. A. Kaufman. Introduction to Volume Synthesis. *Proc. CG International'91: Visualization of Physical Phenomena*, Boston, MA, June 24- 28, 1991, Springer- Verlag, Tokyo, Japan.
5. G.J. Klinker. VDI- A Visual Debugging Interface for Image Interpretation (and Other Applications). *Proc. 2nd Eurographics Workshop on Visualization in Scientific Computing*, Delft, Netherlands, April 22- 24, 1991. Extended version available as technical report CRL 91/2, Digital Equipment Corporation, Cambridge Research Lab, One Kendall Square, Cambridge, MA, 02139, March 1991.
6. R.W. Scheifler and J. Gettys. The X Window System. *ACM Trans. Graphics* 5(2): 79- 109, April, 1986.
7. C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications* 9(4): 30- 42, 1989.