Next-Generation Data Visualization Tools

by

William Ribarsky and James Foley

GIT-GVU-94-27
May 1994

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

7 Next-generation data visualization tools

Bill Ribarsky and Jim Foley

7.1. INTRODUCTION

Scientific data visualization has finally come of age as an important and accepted discipline. While scientists have been using computer graphics to visualize experimental data and computational results for at least 30 years, recent improvements in cost/performance of graphics workstations, more readily available software, and the new-found name "scientific data visualization" have solidified the discipline. Many thousands of scientists regularly use visualizations as part of their work. Our thesis is that scientists are forced to work too hard to create these visualizations, but that the evolving set of visualization tools can greatly reduce the requisite effort. Further, these tools, and those of the next generation, will open significant new ways to understand increasingly complex and large datasets. Principal causes for this will be the increased interactivity that the tools will naturally offer and the need for designs based on careful consideration of scientific and engineering analysis requirements.

One of the keys to success of visualization tools is ease of use. The tools' users – scientists whose interest is finding meaning in data – have neither time nor patience for graphics subroutine packages, graphics jargon, and unnecessary detail. Ease of use can be provided in several ways. First, tools which allow visualizations to be created as easily as with contemporary 2D presentation graphics packages are important, and indeed, several exist. However, the predefined presentations in these packages are fixed and are not readily extensible. The scientist should be able to create new visualizations by interactively binding data values to visualizations, such as three data values to position, one data value to isosurfaces, another to isocurves drawn on the isosurfaces and three more data values to vectors anchored on the isosurface. Such capabilities exist for 2D data presentation: they need to be generalized to 3D.

Going one step further, automatic generation of such visualizations based on characteristics of the data, information needs of the user, effectiveness of various techniques (color, isosurfaces, vectors), effectiveness of various combinations of techniques, and characteristics of the display devices is an appropriate goal for the next generation of visualization

DATA VISUALIZATION
ISBN

systems. Again, some of this has been done for 2D data presentations: the work needs to be extended. Expert system technology can be helpful in making the complex decisions involved here, but human factors and perceptual psychology knowledge are needed if good decisions are to be made.

7.2. WHAT IS VISUALIZATION?

A useful definition of visualization might be the binding (or mapping) of data to a representation that can be perceived. The types of binding could be visual, auditory, tactile, etc., or a combination of these. As Senay and Ignatius (1990) have pointed out, the visual bindings could be further subdivided into sets of marks (e.g. points, lines, areas, volumes) that express position or pattern and retinal properties (e.g. color, shape, size, orientation, texture) that enhance the marks and may also carry additional information. The auditory bindings could be used for iconization or to map data onto elements such as pitch or spatial localization. The tactile bindings could be through force feedback devices, controls with multiple degrees of freedom, and so on.

We can use our expanded definition of visualization to guide us in building very general and extensible libraries of visualization primitives. If in addition we design these primitives with attachable elements and then allow the user to bind data variables interactively to these elements, we have a truly powerful system for visualization and analysis. Such a system will make possible the easy building of mixed visual representations of data (e.g. containing both point glyphs and volume visualizations) that users can customize to best represent their data rather than being arbitrarily restricted to predetermined representations. In addition such systems will prove particularly useful in representing several variables at once and in such a manner that their interdependencies can be made clear. Finally, as we shall show below, the interactive binding coupled with related forms of interactivity will significantly enhance one's ability to analyse complex data.

Next-generation visualization tools need:

a general data model;
effective and efficient graphical representations;
the means to bind from data to graphics.

However, our next-generation data require that we also integrate into our visualization tools:

the means to convert (or filter) data to appropriate mappings;
a general analysis model.

The next-generation data will scale to breathtaking size and complexity. Already molecular dynamics (MD) simulations on the CM-5 model as many as 250 million interacting particles, each changing position over time. On the horizon are simulations of a billion particles or more. These systems are approaching macroscopic dimensions and will provoke

detailed comparisons with continuum models and with experiments. In underwater seismic oil exploration, the data easily number 1 Tbyte. The spatial structures and other variables embedded in these data must be studied in detail and compared with simulations. Finally, in the next 5–10 years, data flowing from the Earth Orbiting System of satellites will total more than 1 Tbyte per day on atmospheric chemistry, air and surface temperatures, winds, cloud patterns, ocean currents and other variables. It will be fruitless to pre-analyse data of this size fully; that will occur on the fly and most likely as a result of what the user has seen in a previous visualization.

Furthermore, comparison of data from different sources in the same visualizations will become commonplace, as Treinish (1991) and others have pointed out. In these circumstances tools for filtering or converting data will be necessary and must be tightly integrated with the visualization process. We call this integrated approach *visual analysis*. In order to make our visual analysis effective, it is necessary to build an analysis model and integrate it with our visualization model. The analysis model should be founded on a set of principles that are applicable in general to scientific and engineering data. We will discuss the building of such a foundation below.

7.3. A VISUALIZATION ENVIRONMENT

What we have described implies the visualization environment of Figure 7.1. If we have a set of data objects and separately have a set of graphical objects, we can delay decisions about how they are interactively joined together (in the binder) until those decisions need to be made – as the data are being viewed and explored. This is the most flexible way to create data visualizations. (We shall see below that it can become *too* flexible, with the result that the user may need expert assistance in choosing the most effective representations.) Furthermore, if we attach a 3D graphical editor to define and group graphical objects (such as glyphs) for subsequent binding, we can put all these modules in a feedback loop that processes through the user's brain to refine the visualization iteratively or to focus on certain aspects of the data.

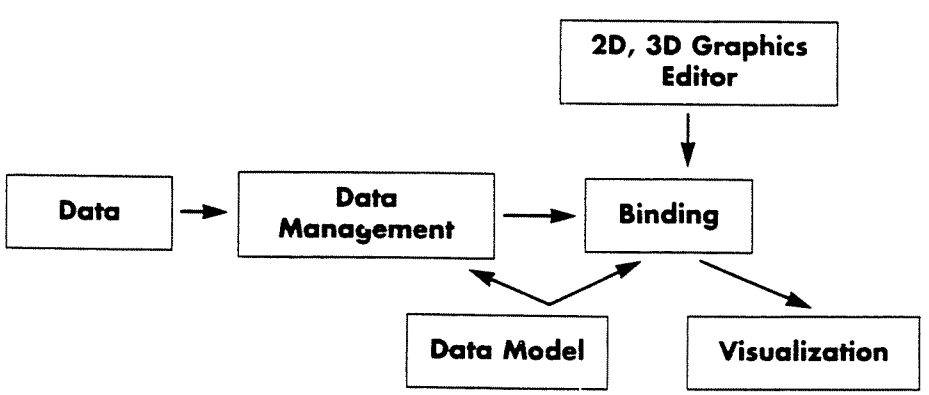

Figure 7.1. A general visualization environment.

It is important that the environment be based on a well-thought-out data model that allows both "expressive" and "effective" data visualizations (in the sense of Mackinlay, 1986). In addition, if the data model is appropriately constructed, the highly flexible binding interface can allow one to distinguish data visually at the level of the individual element (if desired) and can also allow cross-linking of variables to bring out correlations at any level of detail. To permit easy input and use of data from any source, the model should be simple but general and "self-describing", as in Treinish and Gough (1987). Next-generation data will come more and more from several sources and the metadata contained in the data model will be necessary to construct meaningful comparisons. Given the ever-expanding size of next-generation datasets, the data model must be integrated into a carefully developed and very efficient data management structure. We will describe our initial attempts at some of these aspects of a data model in our description of the Glyphmaker.

7.4. GLYPHS AND THEIR USES

Glyphs, which are graphical objects whose elements (e.g. position, size, shape, color, orientation, etc.) are bound to data, have proven useful in depicting spatially complex, multivariate data such as are found in current calculations and experiments. These objects can clearly be effective in depicting discrete data such as macromolecular structure and interactions, but they have also proved their use in representing variables such as wind speed and direction in atmospheric dynamics simulations and observations (Treinish, 1993). Glyphs owe their effectiveness to the ability of the eye–brain system to discern finely resolved spatial relationships and differences in shape. They thus allow the user to display and correlate several variables at once; a researcher can, for example, choose to distinguish two variables by binding them to the visually orthogonal representations of shape distortion and pseudocoloring. We have recently shown (Ribarsky *et al.* 1993a) that complex, three-dimensional structure from molecular dynamics simulations and all components of the stress tensor acting on individual atoms can be distinguished and correlated by using shape distortion and spot coloring of spherical glyphs bound to the atomic positions (see Figure 7.2.). It is important to reveal this interdependence because the atomic stresses force the plastic deformations that result in significant structural changes.

In quite a different context, Ellson and Cox (1988) have shown that glyphs are quite useful in depicting the flow patterns that result from finite-element simulations of hot plastic injected into a mold. In this work 100–200 glyphs were placed according to a finite-element grid; each glyph was a 3D object with the velocity in a finite element represented as a shape distortion of appropriate magnitude and direction, and temperature and pressure represented by different color scales on the glyph and its base, respectively. Other important work employing glyphs concerns the use of color or gray scale "icons" that merge color and texture perception to

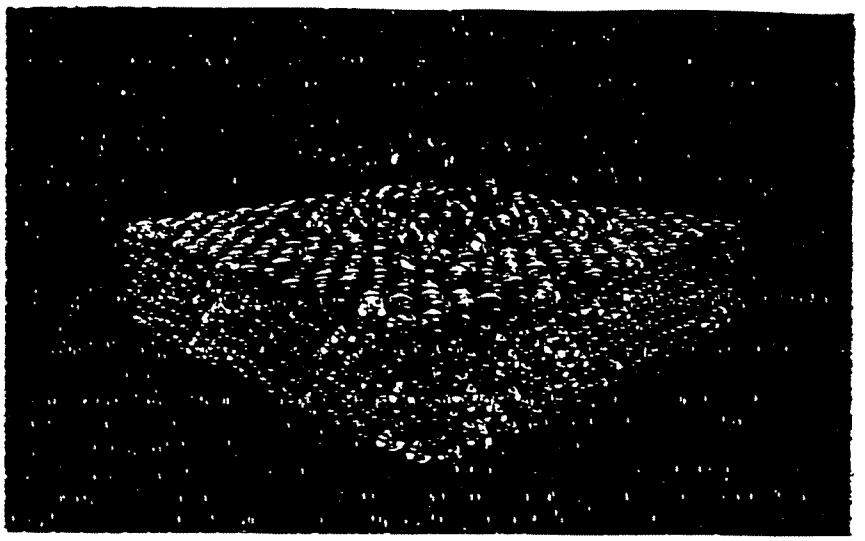

Figure 7.2. Atomic Force microscope with Ni tip being raised from Au surface. On-axis color spots and shape distortion of the spheres are used to represent stretching/compressive forces and shear forces, respectively.

represent multiple parameters (Lefkowitz, 1991; Pickett and Grinstein, 1988). Here the icon is a generalization of the single pixel to objects having perceivable features and attributes. Typical icons are stick figures or other simple collections of lines with such characteristics as line orientation or length representing different parameters. Color coding is also used to bind additional parameters to the icons. The method has been used, for example, on satellite imaging or MRI data (Smith *et al.*, 1991) where thousands of small icons are used to build up collective perceptions of texture and contour.

Some final examples in the current use of glyphs come from work by Haber (1990) on the use of tensor glyphs in visualizing data from engineering mechanics. All these and many other examples show the range and power of glyph representations. They are often superior to traditional surface and volume techniques in, for example, displaying and elucidating 3D vector or tensor fields. It is apparent from this selection of applications, that glyphs can be effectively used to depict a variety of data, including continuum data, in addition to the obvious data (e.g. molecular modeling applications).

Taken together, all this research explored a wide range of glyph representations starting from a few hundred highly detailed glyphs (Ellson and Cox, 1988) and extending to several thousand glyphs where it was most important to perceive "gestalt" or collective effects rather than individual glyph characteristics (Lefkowitz, 1991). Our previous work explored the middle ground where we showed that one could effectively perceive and use (for analysis) individual glyph elements even when there were thousands of glyphs (Ribarsky *et al.*, 1993a). Such an example, where several variables (tensor components) are depicted at once, is shown in Figure 2. With Glyphmaker we hope to make it possible for the user to represent, visualize and analyse data effectively in all these categories.

7.5. AN EXAMPLE: GLYPHMAKER

We have recently developed Glyphmaker, a tool for data visualization/analysis (Ribarsky *et al.*, 1993b,c) that allows users easily to build customized representations of multivariate data and provides interactive tools to explore the patterns in and relations between the data. Glyphmaker is integrated into the SGI Data Explorer to take advantage of the features and flexibility of dataflow visualization systems. The main parts of Glyphmaker are the Data Reader, the 3D Glyph Editor and the Glyph Binder. It provides non-expert users with a quick and easy way to set up visualizations of complex data, to refine the visualization, add to the set of variables depicted and focus on regions of interest. We provide here a brief overview of Glyphmaker; see Ribarsky *et al.*, 1993b,c) for a more complete description with a fuller discussion of design principles and applications.

7.5.1. Foundations of Glyphmaker

Our objective is to provide a highly interactive and efficient means for users to create visual representations, change them, bind data to the representations, change the bindings, bring in new data or glyphs with associated bindings, and quickly see the results of all these operations and updates in a visual display. The historical motivation for our approach comes from work of Foley and others. In a system developed for dynamic process visualization, Foley and McMath (1986) showed how non-programmers could construct and customize dynamically updated scenes displaying process information. This information came from real-world environments such as factories, power plants and refineries and was displayed (and the display modified) to provide precise monitoring of these complex processes. The idea was to make available a library of icons (2D glyphs) or allow the user to create them; the user then employed binding tables to connect variables to icon elements. The user was thus able to create a customized, graphical representation describing a set of time-varying processes variables and to do so without intensive programming. Later Foley (1990) proposed extending this idea to more general visualization schemes, such as with scientific data. Here the data might be 2D or 3D, and the user might need to analyse several variables at once.

The recently developed dataflow approaches provide a general, compatible framework for Glyphmaker. Although dataflow and similar schemes based on a visual programming style had been investigated for a period of years, the first widely-used dataflow system was AVS (Upson *et al.*, 1989). Now there are several additional systems built around the same basic dataflow concept including the SGI Iris Explorer , IBM Explorer, Khoros, and apE . They all allow users to build their own applications by constructing graphical networks of modules – such as for data input, filtering, transformation to geometry, pseudocoloring and rendering.

7.5.2. Glyphmaker architecture and interfaces

The Glyphmaker system has 3 main interactive parts:

the Read Module, which converts input data from a user-specified format to the Explorer internal format so that it can be read-in by any existing Explorer module;
the Glyph Editor, which creates glyphs and arranges them, if desired, into compound glyphs;
the Glyph Binder, which allows interactive bindings or binding changes between data variables and glyph elements. It also allows one quickly to alter the range of data variables or to alter the range of change in the glyph elements as the data varies from minimum to maximum. Finally, the Binder permits the saving of new bindings or reloading of previous ones.

We will describe these parts below. In addition, some Explorer modules are important enough that we integrate them into our system; two main modules here are the Generate Colormap module and the Renderer module. Figure 7.33 shows how the parts of Glyphmaker fit into the Explorer dataflow environment. Note that the structure in Figure 7.3 fits into the general structure shown in Figure 7.1. Data management is handled through the Data Transcriber and the Header/Label Transcriber, and the data model is organized around the Explorer data structures and the Header/Label file.

The Read Module

To deal with the need to retrieve ever-growing amounts of data from a variety of formats and place them in a common structure for easy comparative viewing or for analysis with a common set of tools, we have developed a simple "self-describing" data structure that is consistent with other work on this problem (Treinish and Goettsche, 1991). Each self-describing file has a header, which describes the data and their format, followed by the data values themselves. In the header each variable is labeled with a name, primitive type, number of instances, minimum value and maximum value. The label, minimum and maximum values are important information that is passed along to the Glyph Binder. The number of records following in the file with identical format is indicated at the top of the header; this allows concatenation of an animation sequence or different views of the data. Each header can also contain comments about what the file contains. The data in the file then follow in the order given in the header. Headers (and the data following) can be placed anywhere in the file. They can also be placed together at the top with all the data following the last header; this allows one to merge files and still keep header information accessible.

We chose this very simple format because it was easy to understand, easy to create, and it fulfilled the basic needs of the Glyphmaker. In the future we will modify the Read Module to handle more complicated data formats

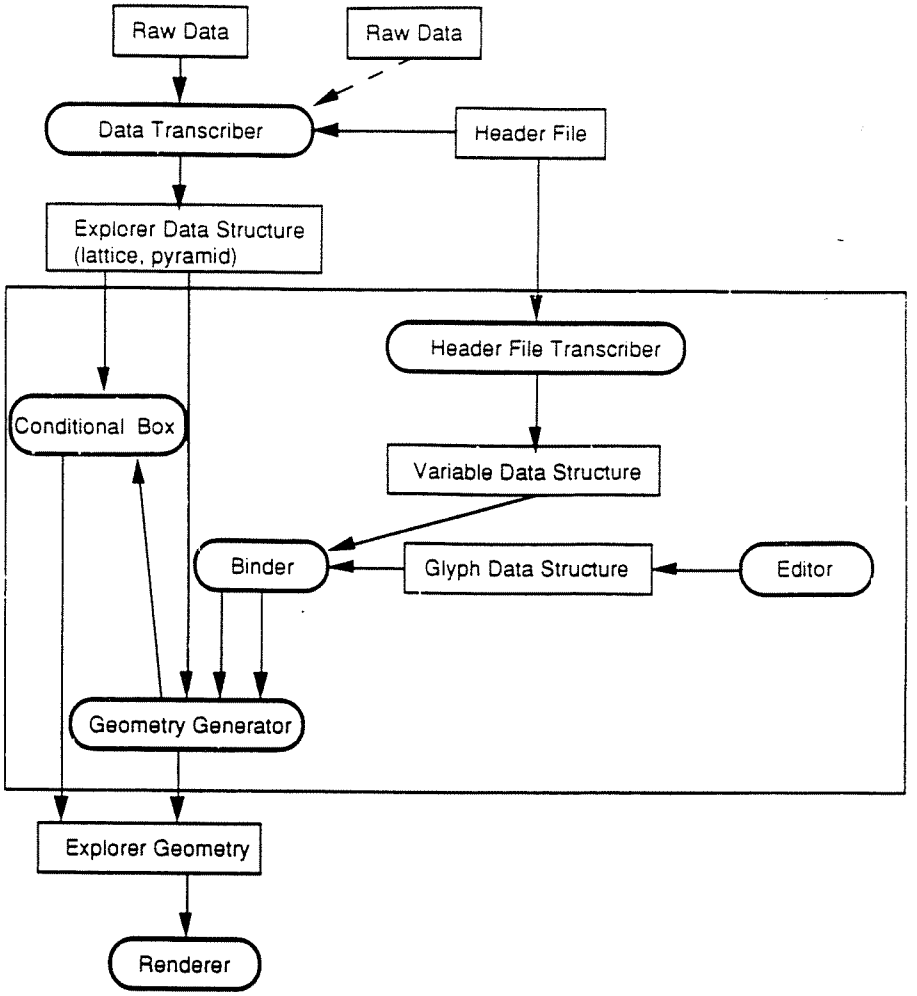

Figure 7.3. Glyphmaker visualization environment. The figure shows the flow of data between the parts of Glyphmaker as described in the text. Raw data flow into the Data Transcriber (in the Read module), are converted to Explorer data structures, and then used to generate geometry. The Header file (which can be part of the data file) is used by the Binder to make data variable headings that are joined with glyphs from the Editor.

than described here, but we will try to do so while retaining the basic simplicity and intuitive feel of our present format.

The Glyph Editor

The Glyph Editor is a simple 3D editor used to draw the glyphs and, if desired, to arrange and orient them in compound glyphs. There are four selection menus, seen in the upper right hand corner of Figure 7.4: Glyphs, Views, Mode, and Action. The Glyph menu items are: lines, spheres, cuboids, cylinders and arrows. The Views menu helps the user manipulate the glyphs in 3D space by providing menu items for selecting either perspective, top, side or front views singly or all four views

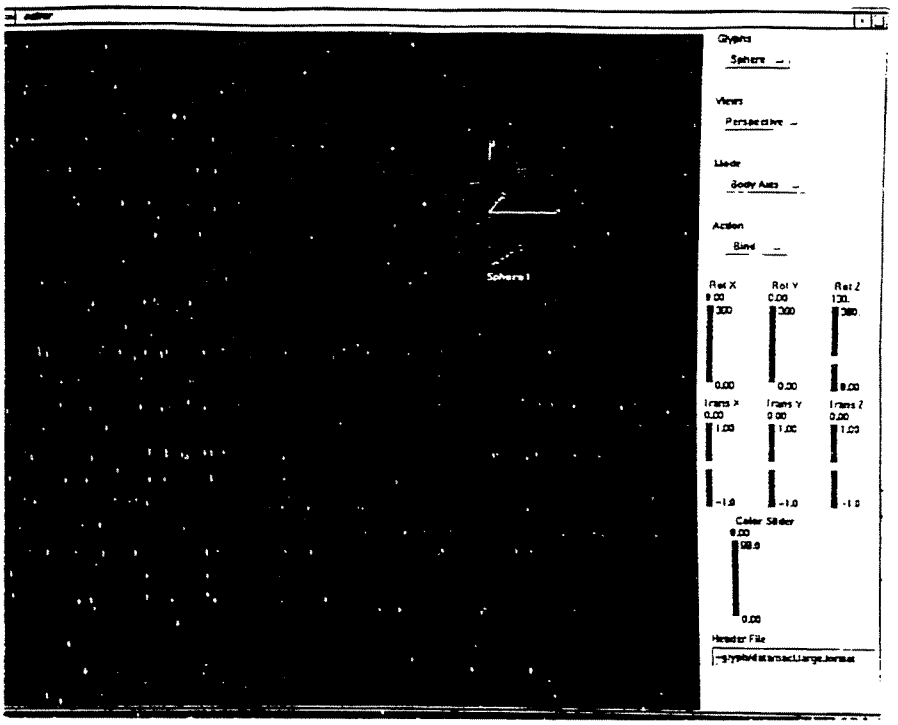

Figure 7.4. 3D Glyph Editor interface. Selection menus are in the upper right corner; a compound glyph composed of a sphere and a vector glyph is displayed.

simultaneously. The Action menu has bind, render, ungroup and clear items for (respectively) bringing up the Binder, rendering a set of glyphs (with full lighting and shading – useful for seeing how they will look in the finished visualization), ungrouping grouped glyphs, and clearing all glyphs from the Editor. We will soon include options to save and load useful glyphs, since creating good glyphs takes care and testing.

Each glyph has several elements that can be bound to data. For example, one might bind the radius, position (x, y, z), color, overall size and transparency of a sphere to individual data variables. Shape, such as elongation or shortening along each axis, might be bound to additional variables. Similar sets of elements are available for all glyphs in the Glyphmaker library. One must choose which elements to make active for the next phase; only these elements appear in the Glyph Binder menu.

The Glyph Binder

For binding, the user chooses the bind item from the Action menu in the Editor. A window pops up that shows all the active elements and the variables as toggle buttons (see Figure 7.5). To bind a variable to an element, the user simply selects an active element followed by a variable or a variable followed by an active element. As shown in Figure 7.5, the user can also get a pop-up window with binding information by clicking an already-lit toggle button. Bindings can also be deleted and reset with the mouse. Attached to each variable and active element are text widgets

specifying the minimum and the maximum value for that variable or active site. Default values appear initially in these widgets – the default for the variable is taken from the self-describing file header. The use of defaults means that bindings can be established in a few clicks, and the results displayed immediately. Of course, the user has the ability to change any of these values, and the result will be transmitted to the renderer for update of the display.

We have tried to create the Binder with the interactivity and ease of control necessary for effective data exploration. The variables and elements are in separate scrollable lists (Figure 7.5) so that one can quickly find the variables/elements of interest, even if the lists are longer than the window size. The Binder window is also resizable and movable using the mouse so that one can easily rearrange the display real estate – this is important since the Binder will often be used concurrently with the renderer for interactive refinement of data visualizations. The Render button at the bottom of the window pipes any changes made in the Binder to the renderer for display. The OK button dismisses the Binder, but with data/glyph bindings remaining in place for the interactive visualization. At the top left of the window, the File button has a pull-down menu with the usual save/reload options. Saving and loading of bindings is important since it may take some time to find an ideal set of bindings and associated minimum/maximum values for a particular set of data In addition to the local presentation of binding information obtained by clicking on a particular toggle button, the user needs a global view – especially when several glyphs are involved. To handle the global view, we are near completion and will soon implement a binding visualization panel in the Binder window that will illustrate the overall binding structure.

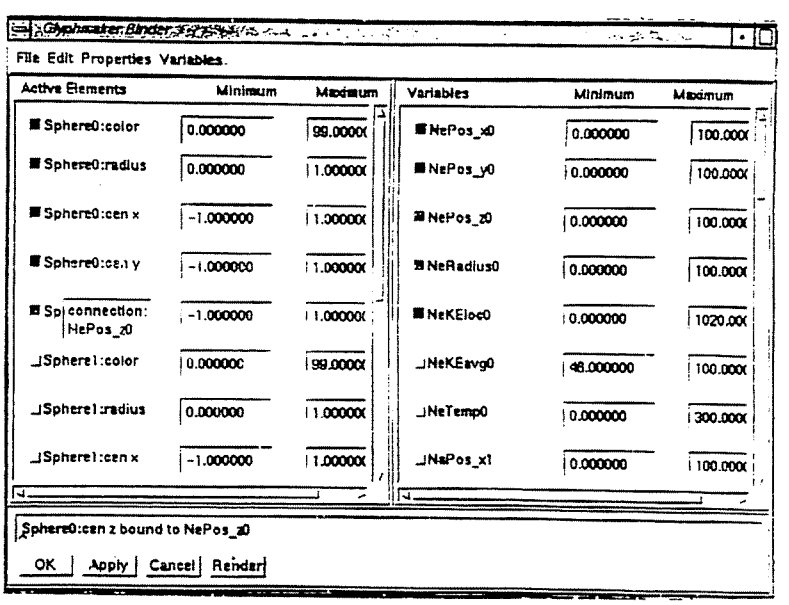

Figure 7.5. The Glyph Binder Interface. On the left are the glyph active elements and on the right are the variable names (in this example including position coordinates and properties for each atom type).

Conditional Box

We have implemented a specific and particularly useful instance of a concept from statistical graphics called *focusing* and *linking* (Buja *et al.*, 1991). (As discussed further below.) This is a 3D box that can be moved and sized to enclose a subregion of the data after it has been selected for display and bound to particular glyphs. The box allows one to distinguish what is inside from what is outside – in effect to reclassify the data according to this condition. Thus the data, which may have been originally classified according to atom type can be additionally classified according to this spatial condition. One can then choose new glyph bindings – for example, new colors or glyph shapes for what is inside the box – to focus on or even isolate an interesting spatial region.

Our current implementation of the Conditional Box is shown schematically in Figure 7.3. The box module receives information from the Explorer data structure and from the Geometry Generator to generate a box with the appropriate scale and position with respect to the data. Then data and box geometry are piped to the renderer for simultaneous display. Interactive controls can update the box geometry and position it with respect to the data. In the future we will pipe the new classifications determined by the Conditional Box (e.g. in or out) directly to the Binder, allowing the user to employ the full range of glyph elements. At the moment we only allow a limited set of glyph rebindings based on the Conditional Box.

A Glyphmaker example

We present an example using Glyphmaker on data from a molecular dynamics (MD) simulation investigating the dynamic behavior of atomistic materials as they move and interact with one another. External and internal forces may cause the build-up of large, localized and transient stresses and heating resulting in such dynamic events as plastic flow, localized melting and resolidification, and even rupture or splattering of material. It is necessary to analyse the complex interrelation of stresses, heating and dynamic behavior in order to understand the processes that occur in these simulations. The simulation presented here was run on a Cray Y-MP at the DOE National Energy Research Center by Dr Hai-Peng and Professor Uzi Landman of the School of Physics, Georgia Instutute of Technology (Hai-Peng and Landman, 1993).

The simulation depicts a "nanocluster" of 64 atoms of NaCl hitting an Ne surface and becoming embedded in the surface. Although the system was initially quite cold (50 K) and the impact velocity moderate (3 km/s), upon impact the cluster and surface in its vicinity heats up greatly with much disorganization and eventually some vaporization. We have chosen simple sphere glyphs for binding to the separate components of the system (Na and Cl separately in the cluster, dynamic Ne atoms in several layers stacked on several layers of crystalline substrate). Properties of this system include localized atomic kinetic energy and temperature and also the atomic stress tensor – in the visualizations presented here we have bound the temperature to the color of the glyph for each atomic type. We have applied the Conditional Box to the system at a time step just before the

cluster penetrates the surface (Figure 7.6). This is to avoid the common problem with large scale simulations; often important features are embedded in the system and hidden from view, as shown in Figure 7.7 where the cluster has fully penetrated the Ne (with binding of color to local temperature turned on). Using the interactive rotation and zooming capabilities in the renderer and the Conditional Box controls (shown at the lower right in Figure 7.6), we adjust the semi-transparent box so that it encloses the cluster and the region of the Ne surface that the cluster is about to strike. Now we can choose to view only this selected region, the excluded region, or both (perhaps using the transparency control to make the excluded region semi-transparent). We can also retain the Conditional Box classification shown in Figure 7.6 for later time steps; Figure 7.8 shows the selected region after the cluster embeds itself (with local temperature binding turned off and on, respectively). In this case, we can see (by comparing with previous time steps) that Ne atoms have diffused out from the center of the impact but the edges of the selected region have not yet moved out much. Finally, we show in Figure 7.9 the full system with the cluster removed (achieved by just clicking off the NaCl bindings and updating the display). One can now explore the shape and development of the hole using the interactive rotation and zoom controls in the renderer. Also, though it was not apparent when the cluster was visible, one can see clearly that the greatest heating of the Ne surface remains in a concentrated region below the cluster impact. The color bar in Figure 7.9 shows the temperature binding used throughout. A temperature range of 0 to 300 K is mapped onto the absolute scale of 0 to 100 on the color bar. We can thus see that even after total embedding of the cluster, the outer edges of the surrounding Ne remain quite cool. However, the cluster itself and Ne atoms in the center are between 200 and 300 K – much greater than the vaporization temperature of Ne.

In Figures 7.10 and 7.11, we show how more properties can be built into the visualization by depicting the effect of localized, on-diagonal atomic stresses using a vector glyph. The same time step as in Figure 7.7 is shown, but here we have removed the binding to the Ne layers and enabled the binding to the crystalline substrate atoms below so that we can see clearly how the latter behave. Since the stress tensor is symmetric, it has only six components that can be divided intuitively into on- and off-diagonal sets. The on-diagonal set represents stretching or compression along the component axes depending on whether the sign of the component is positive or negative, respectively; the off-diagonal set represents shear forces. We have depicted the on-diagonal forces both with a grouped glyph consisting of a sphere and vector (not shown here, the spheres locate atoms precisely) and with vectors alone. One can switch between the two representations with a few clicks in the Binder. The direction and magnitude of the vector indicates the amount and direction of on-diagonal force. (The magnitude of the vector is also proportional to the pressure at that point). Let us label the horizontal and vertical directions in Figure 7.11 as x and y with z perpendicular to the image. Figure 7.10 shows that columns of atoms (in the z direction) directly below the cluster (and below rows along x and y that intersect these atoms) have vectors mostly pointing downward indicating compressive forces along z. However, Figure 7.11 shows that columns that

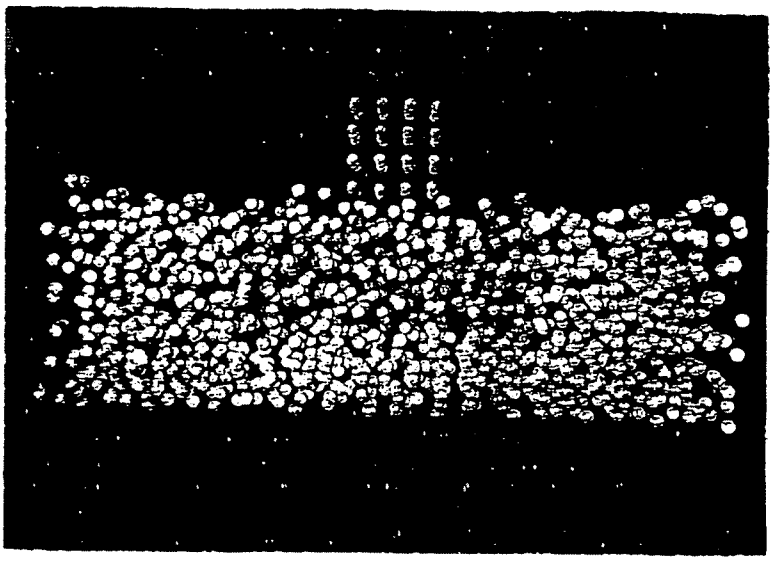

Figure 7.6. The full MD system with the semi-transparent Conditional Box surrounding the NaCl cluster and Ne in the impact region.

are more along the diagonals from the center of the image have larger components along the crystalline surface plane. This crystal-structure dependent distribution of stresses is brought out by the Glyphmaker bindings. One could also represent the off-diagonal stresses by adding another vector, or the stress tensor could be represented in a variety of other ways. We work out the application of Glyphmaker to tensor quantities in Ribarsky *et al.* (1993c); we also present a completely different application there, visualizing a computational fluid dynamics simulation of airflow around helicopter blades.

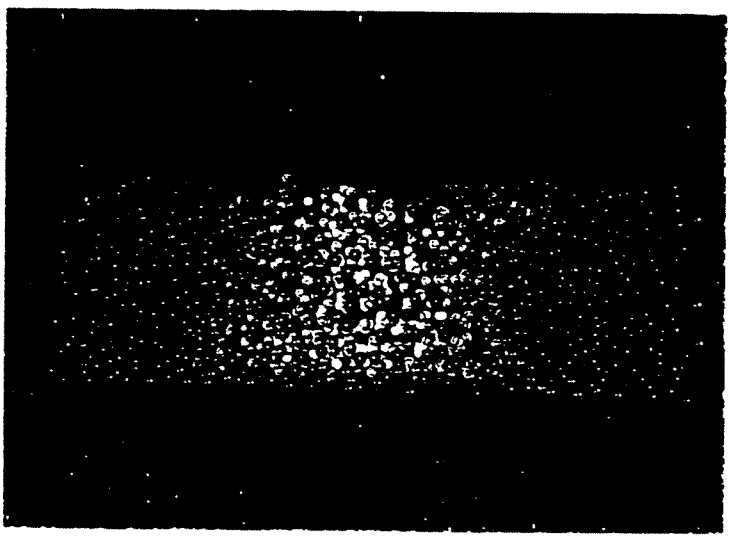

Figure 7.7. The full MD system at a time step after the cluster has embedded itself in the Ne; here the color glyph elements are bound to local temperature.

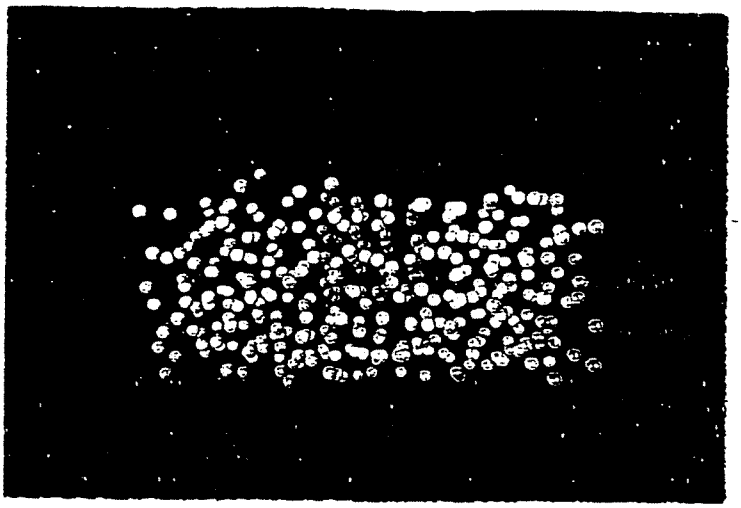

Figure 7.8. The part within the Conditional Box at the same time step as Figure 7.7.

7.6. EXTENDING GLYPHMAKER

It is evident that the Conditional Box offers a general means to control the data visualization. If the Conditional Box is used to provide a new set of class labels (by sending the new label information to the Header/Label Transcriber in Figure 7.3), then bindings of glyphs to data can be interactively reset, based on these new classes. This provides the power to bind at any level of detail, even down to the individual datum, that we referred to

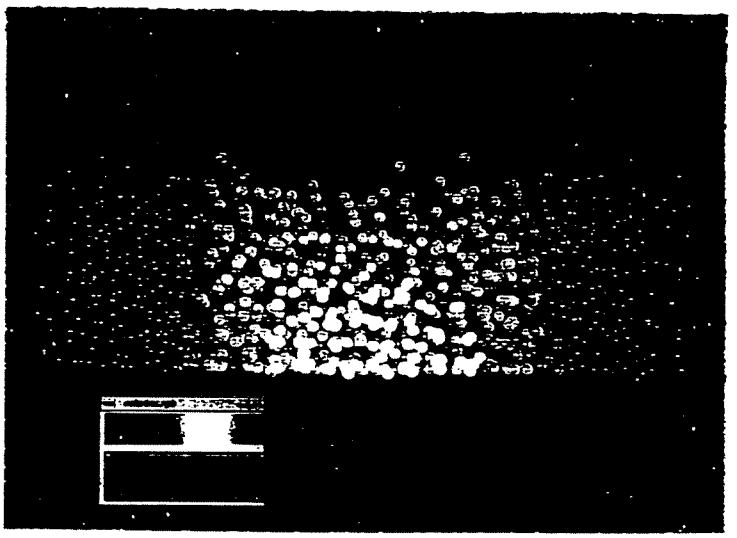

Figure 7.9. The full MD system (same time step as Figure 7.7) with the NaCl cluster bindings turned off and with color bound to local temperature. The color bar for the local temperature binding is in the lower left corner; the 0 to 100 scale maps linearly onto 0 to 300 K.

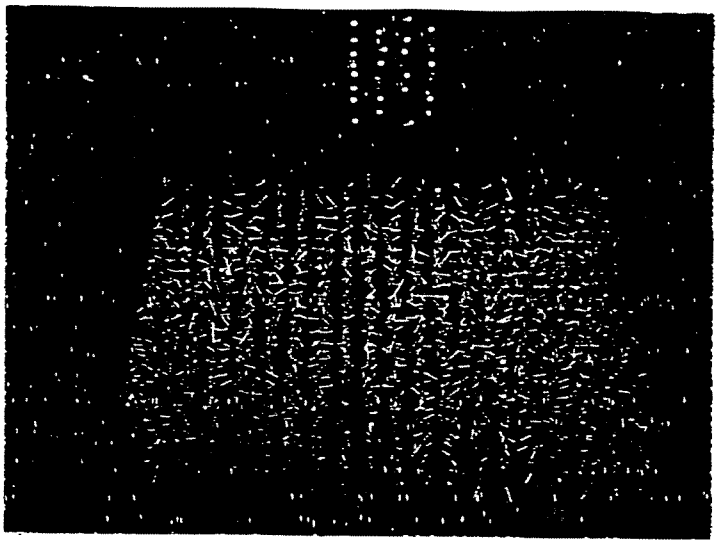

Figure 7.10. The NaCl cluster and crystalline substrate with Ne bindings turned off (same time step as Figure 7.7) with visualization of the on-axis (stretching/compressive) components of the atomic stress tensor added.

previously (e.g. the spatial Conditional Box can be set to any size). Of course, the Conditional Box need not only be applied spatially; it could be applied to any property variable. Thus one could set classes based on ranges of energy, temperature, force components or other variables. If a mechanism is provided to save and recall each of these classifications, they could be collected for comparative viewing. In addition, there is no reason to restrict the user to just one Conditional Box; he or she should be able to generate multiple boxes and choose several ranges, if need be. Finally, of course, one should be able to apply conditions more complicated than the selection

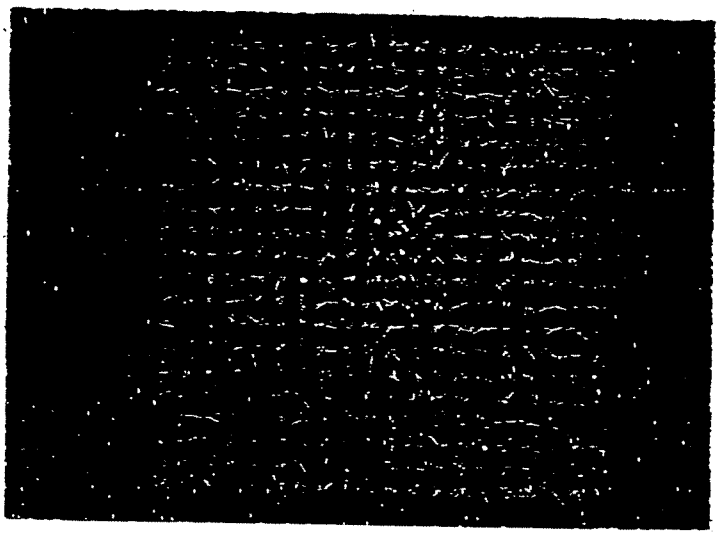

Figure 7.11. A rotated view of the system in Figure 7.10. The viewpoint is directly over the NaCl cluster and perpendicular to the crystalline substrate surface.

of variable ranges. These might involve algebraic expressions for data variables or correlations between two or more Conditional Boxes.

A common thread in all these extensions is that the Conditional Box provides the means for *focusing* and *linking* to bring out correlations between data. These are processes that have proved quite successful in statistical graphics where multiple views, each containing partial information about a complex dataset, are either isolated or linked (e.g. perceptually) so that the user can integrate them into a coherent picture of the data relations (Buja *et al.*, 1991). In fact, if we pipeline the reclassifications of the Conditional Box to a predetermined set of bindings and then move the box interactively, we have a version of the *brushing* techniques as well (Buja *et al.*, 1991). We are working on extending Glyphmaker to make many of these Conditional Box operations possible. The advantage of the Glyphmaker approach is that one will be able to apply a condition to the data, see the visualization updated based on the condition and adjust accordingly. This capability for iterative refinement will produce optimized visual analysis.

7.7. 0D, 1D, 2D, 3D, OR BEYOND?

Glyphmaker and the other glyph-based examples mentioned above display zero degree of freedom (0D) data, since icons describe points and their associated properties. How can we handle higher dimensional representations (e.g. ribbons, isosurfaces, volumes)? Our examples and those of other workers have shown that 0D glyphs can be effectively applied to a broad range of data types, including continuum data. Further, it is apparent that point glyphs have certain advantages over other representations. They can, for example, be applied to original (even noisy) data and do not require the amount of interpolation and smoothing that, say, surface representations impose. Point glyphs could also be more effective in bringing out strong variations that may exist in all three directions in spatially dense data since higher dimensional representations could obscure interior data or not show all variations at the same time. Finally, point glyphs are especially effective, as we have shown, in carrying and displaying several variables simultaneously.

Nevertheless, despite the flexibility of point glyphs, there are cases where other representations are more effective. For example, ribbons are best for showing twisting spatial structures or flowlines that vary in time. Surfaces or volumes may be best for bringing out the general features and patterns in certain continuum data, especially data that are fairly smooth. Even though some of these higher dimensional representations already exist in the dataflow environment, it would be advantageous to put them in Glyphmaker to make use of the interactive binding and conditional focusing and linking that we have come to value. Also, our above example shows that it will be possible (especially as we enrich the set of glyphs) to define unique representations based on these forms that are customized for the data at hand.

How then can we put the higher dimensional representations in Glyphmaker? We can certainly take a cue from spreadsheets, which allow the

interactive binding of datasets to lines, columns, piecharts, etc.. The concept we have followed so far is "one-to-one" binding of data objects (points plus associated properties) to glyphs. (Actually we also allow "one-to-many" since the Glyphmaker binder allows any number of glyph elements to be bound to one variable.) We need to extend this to include the concept of "many-to-one" (i.e. many data objects to one glyph). For example, the glyph might be a ribbon with associated elements of color, width, or a set of transverse spatial dimension that are orthogonal to the ribbon trajectory. The latter, along with color, have been used by Delmarcelle and Hesselink (1993) to depict continuous tensor fields. The appropriate data in this case would be a collection of data objects along a trajectory. Thus whether the glyph is a ribbon, surface or volume it implies a particular data structure for the data objects and also a data model that allows the handling of the glyph in the same way in the editor and the binder, no matter what its dimensionality. Beyond that a particular data preparation is also implied. It does not make sense to bind data to a ribbon if they are not processed as a trajectory (e.g. computed as a streamline or hyperstreamline in the case of CFD). The best way to ensure that trajectories are available when needed is to provide the facility to compute them from more primitive data within the visual analysis system. This approach fits the analysis model, expanded upon in the next section, that we argued must be integrated with the visualization tools. Similar considerations hold for surface and volume data. In principle it appears that a data model and an appropriate set of higher dimensional glyphs with elements can be developed and inserted into Glyphmaker. This will provide the glyph-based approach with the widest possible set of visual representations for depicting all types of data.

7.8. ELEMENTS OF VISUAL ANALYSIS

We have argued for an integration of visualization and analysis because next-generation data will be so numerous and heterogeneous that they must be analysed on the run, and because of the great advantages offered by interactive visualization techniques in this process. In order to establish a general approach, we present here certain essential elements. There is not space to expand on these ideas, but we are working towards a fuller exposition to be presented elsewhere.

For present purposes let us focus on data likely to come from scientific and engineering applications. Clearly a general analysis approach should be based on a study of the general characteristics of the physical quantities associated with the data; that is, characteristics reflected in all (or most) physical data. For example, much of the data come in the form of either scalar, vector, or tensor fields that depend on space and time. These fields have groupings in 3^0, 3^1, and 3^2 components, respectively. In general, 3^n dimensional fields are possible, and all these quantities imply the primacy of the spatial coordinates in the physical system since they are usually derived from expansions of source fields in the spatial coordinates. These groupings imply necessary groupings for visual analysis – if possible, we would want to look at all three vector components or all nine tensor

components in the same visualization. The glyph-based work and other research mentioned above have already provided several methods to do this effectively. However, to be certain that we have a rich enough set of visual representations, we should make a general study of likely physical quantities. A good start has already been made on this for certain classes of physical systems (Delmarcelle and Hesselink, 1993; Haber, 1989), and there are general studies that can be applied (Borisenko and Tarpov, 1979). Beyond this question, there may also be other "necessary multidimensional groupings" that are not 3^n. It is necessary to see whether the representations (or modifications of them) we have built for the spatial fields will be effective for these cases and what additions to our visual vocabulary may be needed.

For quantitative analysis, we need an algebra for combining variables that reside in our data objects. This certainly should include a standard set of functions (log, exp, root, power, etc.), generally useful transformations (linear algebra operations such as diagonalization, etc.) or statistical techniques (regression analysis, smoothing, etc.) . Perhaps a command line interface would be the most flexible for these operations. In any case, our data model must be flexible enough to handle these transformations efficiently and produce new data objects of the appropriate type for further visual analysis.

Finally, we should address the deeper question of how researchers tend to analyse and then develop understanding of complex data. For example, they often reduce the dimensions of complex results by averaging over one or more variables (including time). They then study these simpler quantities and develop conceptual models that may lead to analytic models. If the data come from observations, this may eventually lead to more complete analyses, based on simulations or multivariable phenomenological models, where the whole physical system is studied. This sequence of investigation suggests that we should insert ways to choose "moments of the data" in our interactive visual analysis system. Obvious moments could be selected by, for example, putting boxes around a spatial region of the data and then averaging over a chosen spatial dimension. The researcher could then follow the length, width or height of this region in time. Of course, extensions of this idea could be made to other variables. Certain data may also exhibit periodic behavior or other patterning in space, time or other variables. Fourier and related transforms should be available and applicable interactively to reveal this behavior. This is often quite a powerful way to provide a unique signature for the data (analogous to the structure factors from crystallography) that may be revealed in only a few dominant components. These signatures may suggest causes for the patterns in the data or their dynamic behavior.

7.9. TOO MUCH VISUALIZATION EXPERTISE REQUIRED

The preceding direction for visualization software concerns rapid end-user creation of the data visualizations. Unfortunately, even with the best currently-known tools (in our opinion, a combination of dataflow

diagramming and glyphmakers), the scientist seeking to visualize his information still has to know too much. Going beyond these tools, we want to help the scientist choose an appropriate visualization without the need to try out a large number of possibilities. Consider the case of a dataset with ten 2D scalar fields, and with eight ways to depict a scalar field (hue, saturation, value, isolines, projection into 3D, size of a glyph, orientation of a glyph, elongation of an elliptical glyph). Without considering the combinations of these display methods which do not make sense, there are 10^8 ways to bind the scalar fields to the visual representations. How is the scientist to decide which to choose? There is really no practical way to decide, other than to make the same choices that others have made in the past, or to experiment painstakingly with a small number of the possibilities. This is an unsatisfactory situation.

What we really need are tools which allow the scientist to work at a higher level of abstraction than currently. The scientist should be able to say to the computer "I would like to see a comparison between the stress tensor and temperature fields from time t_1 to time t_2". The computer should then propose to the scientist one or several possible visualizations which meet that request.

What we are talking about here is declarative rather than procedural specification of what we want from the computer. Rather than telling the computer *how* to get what we want, we simply tell the computer *what* we want, and expect the computer to deduce how to get here from there. This is exactly what is done in the database world, where a typical query might be "retrieve all data files concerning ozone levels over the north pole between 1970 and 1990 for which the average ozone level is greater than 50 metric tons per square mile below the seasonal norm".

The current solution to this problem is to have the scientist talk not with the computer but with a "visualization designer" who understands graphic design, has perception, and has some knowledge of scientific concepts and needs. NCSA calls this the "Renaissance Team" approach (Smarr, 1988). This can work in the short term, but in the long run, it is neither practical nor economical to provide each scientist with easy access to a graphic designer. Nor is it practical to train scientists to become expert visualization creators.

The only viable longer term solution is to imbue visualization tools with some graphic design expertise. This could be done by having the computer serve as a critic of what has been designed by the scientist, or by having the computer actually generate one or several possible visualizations. The first approach, acting as a critic using the stated visualization objective, still leaves the creation process to the user. The automatic generation alternative is the more attractive, although also the more difficult: it is always easier to criticize an artifact which someone else has created than to create the artifact in the first place.

There are at least two general approaches to automatic generation, both developed in the artificial intelligence community: the classical *generate-and-test* and the more recent *case-based reasoning*.

Generate-and-test. The generate-and-test strategy requires a set of graphic primitives, rules for generating visualizations from the primitives, and criteria against which to test each generated visualization. For instance, in

the natural language domain, words form the primitives of the English language, grammar rules tell us how to generate sentences from words, and our knowledge of the real world is the criterion by which we test a sentence to determine if it makes sense.

In the graphical domain, Mackinlay (1986) was the first to use the generate-and-test paradigm. He worked with 2D discrete data presented in business-graphics formats such as bar, scatter and line charts. His graphics primitives include areas (such as circles and bars), lines, marks (such as + and °) and visual attributes such as color, size and orientation. The primitives are organized into *primitive graphical languages*. Each of these primitive languages uses one or several graphics primitives. The primitive graphical languages include: horizontal axis, vertical axis, line chart, bar chart, scatter plot, color, shape, size, saturation, texture, orientation, tree and network.

Small open circle not degree

The grammar for generating alternative representations is defined by three composition rules, which combine the primitive graphical languages into complete graphs. The rules are double-axis composition, single-axis composition and mark composition. Double-axis composition superimposes two sets of data encoded with identical vertical and horizontal axes on a single set of axes, such as showing the sales data by year for two different divisions on the same set of axes. Single-axis composition juxtaposes two sets of data which share one common axis (but have a different second axis) such that the common axis occurs just once. An example here would be two bar charts side by side, one encoding gross national product (GNP) by country and the other encoding population by country. Country is the common axis, while GNP and population are the different second axes. Mark composition uses two different attributes of the same area or mark to encode two different variables, such as using the size of a circle to encode a state's population and the color of the circle to encode the average age of the state's population.

Rules for testing the generated alternatives are derived from a combination of experimental studies (Cleveland and McGill, 1984; Cleveland, 1985; also Ware and Beatty, 1988, the published version of earlier technical report used by Mackinlay) and experience-based guidelines (Bertin, 1981, 1983; Tufte, 1983). The rules concern what types of information (nominal, ordinal, quantitative) can be expressed by different primitives, (example) and the relative effectiveness of alternative primitives which could be used for the same type of information. For instance, size and saturation can both be used to express quantitative data, but size is more effective than is saturation.

Senay and Ignatius (1990, 1991, 1992), working with Foley, Hodges and Sibert, extended Mackinlay's conceptual framework to 3D visualization. Additional composition rules are defined: in addition to double-axis composition, single-axis composition and mark composition, Senay and Ignatius define superimposition (placing one set of data on top of another), composition by transparency (making one set of data partially transparent on top of another), composition by intersection (using the 3D intersection of two sets of data to depict both sets), and composition by interleaving (spatially interleaving marks encoding one dataset with marks encoding another). Figures 7.12 and 7.13 illustrate some of this work.

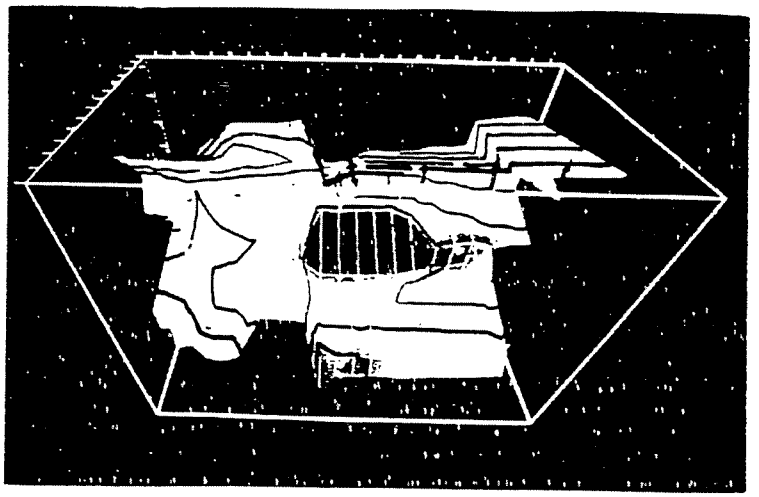

Figure 7.12. A VISTA-created visualization created in response to a request (created via dialog box fill-in) to show weather data variables from Peru, in decreasing order of importance: rainfall, shown as a surface (because this is the most effective available technique for showing continuous 2D data; humidity, shown as contours on the surface (a less-effective technique than the surface, chosen because the user stated that humidity was less important than rainfall), temperature, shown as color (even less effective than the contours); wind velocity, shown as blue vectors; and observation data location, shown as the small white triangles. The composition operator of superimposition has been used to combine the contour plot and station locations onto the surface depicting rainfall. (Courtesy H. Senay.)

Roth and Mattis (1990), in their SAGE system, extend Mackinlay's work in another way, adding more semantic description to the data and their relationships. For example, they represent algebraic relationships such as total–costs = materials–costs + labor–costs, allowing them to reflect this in the graphs they produce, such as a bar chart showing total–costs but with separate bar segments for the bar showing material–costs and labor–costs.

Casner's (1991) BOZ takes a different approach, focusing first on the user's objective, or task, in requesting the visualization. Different visualizations are generated. For each visualization, the sequence of visual search and comparison operations the user will have to perform to meet the stated objective is also generated. The test criterion, used to determine which of the generated alternatives to accept, is essentially the time which will be required for the user to perform the sequence of operations. Casner experimented with visualizations generated by BOZ and found that users' performance was indeed better than with less well-designed visualizations, and that the predicted task time correlated well with the measured task times. Using the generate-and-test design strategy, the sequence of visual scanning and comparison operations needed to achieve the objective with different potential visualizations is used as the criteria for testing each of the alternatives.

Generate-and-test has been well-explored, and shows considerable promise. Its application depends on formalizing the generation process and

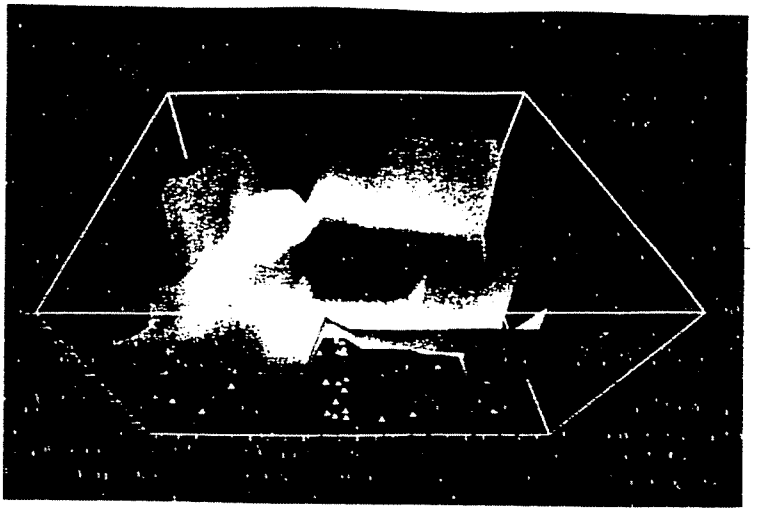

Figure 7.13. A second visualization of the same weather data, in which the supermposition operator has been replaced by what Senay calls union and Mackinlay calls double axis composition. The result is that the humidity contours, wind velocity vectors, and station locations are on the ground plane rather than on the rainfall surface, creating a less effective presentation because some of the information is obscured. (Courtesy H. Senay.)

quantifying the test metric: for more complex visualizations, or new domains, this is a non-trivial and not always possible task.

Case-based reasoning. In contrast to the generate-and-test strategy is case-based reasoning (Kolodner, 1993). The fundamental strategy is to organize a large collection of existing visualizations as *cases* and to design new visualizations by adapting and combining the past cases. New problems in the case-based approach are solved by adapting the solutions to similar problems encountered in the past. The technique has been used in a wide variety of applications. It has not yet been applied to scientific data visualization, but we believe it holds promise as an alternative to generate-and-test because it requires less formalization.

The important issues in building a case-based visualization advisor are developing a large library of visualizations, developing an indexing scheme to access relevant cases, and determining a closeness metric to find the best matches from the case library. We are exploring this area, but have not yet initiated serious work. The idea would be to index the cases by the types of data to be presented (nominal, ordinal, interval, scalar), the relative importance of each of the datasets, the dimensionality of the data space (typically 2D, 3D or 4D, with time possibly being one of the dimensions) and the tasks to be performed with the visualization (such as compare, find relative change, correlate, determine value). For purely exploratory data analysis, of course, stating of a goal could be difficult.

Whether generate-and-test, case-based or other approaches are used, it is also important that the visualizations generated or retrieved by the system be editable by the user. Creating a system environment in which the system

presents the user with the "best" or "right" design is to make the arrogant presumption that the system is always going to do at least as well as the user could do – an unlikely proposition. We know how important this flexibility is from our work with automatic generation of user interface dialog boxes and menus (de Baar *et al.*, 1992; Kim and Foley, 1993). In the context of scientific data visualization, an attractive way to do this is to generate the dataflow diagram and parameter settings needed to create the visualization: then the user can fine-tune the visualization with a well-known set of tools.

ACKNOWLEDGEMENTS

This work is supported in part by grant numbers CDA-9200635 and NCR-9000460 from the National Science Foundation.

REFERENCES

Bertin, J. (1981). *Graphics and Graphics Information Processing*. de Gruyter, New York. (Translated by W. Berg and P. Scott from *La Graphique et le Traitement Graphique de l'Information*, Flammarion, Paris, 1977.)

Bertin, J. (1983). *Seminology of Graphics*. University of Wisconsin Press, Madison, WI. (Translated by W. Berg from *Smiologie Graphique*, Editions Gauthier-Villars, Paris; Editions Mouton & Cie, Paris-La Haye; and Ecole Pratique des Hautes Etudes, Paris, 1967.)

Borisenko, A. I. and Tarpov, I. E. (1979). *Vector and Tensor Analysis with Applications*. Dover Publications, New York.

Buja, A., McDonald, J. A., Michalak, J. and Stuetzle, W. (1991). Interactive data visualization using focusing and linking. In *Proceedings of Visualization'91*, San Diego, pp. 156–163.

Casner, S. (1991). A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics* **10** (2), 111–151.

Cleveland, W. S. and McGill, R. (1984). Graphical perception: Theory, Experimentation and Application to the Development of Graphic Methods. *Journal of the American Statistical Association* **79** (387), 531–554.

Cleveland, W. S. (1985). *The Elements of Graphing Data*, Wadsworth Advanced Books and Software, Monterey, CA.

deBaar, D., Mullet, K. and Foley, J. (1992). Coupling Application Design and User Interface Design. In *Proceedings CHI'92 – SIGCHI 1992 Computer Human Interaction Conference*, ACM, New York. pp. 259–266.

Delmarcelle, T. and Hesselink, L. (1993). Visualizing Second-Order Tensor Fields with Hyperstreamlines. *IEEE Computer Graphics and Applications* **13** (4), 25–33.

Ellson, R. and Cox, D. (1988). Visualization of Injection Molding, *Simulation* **51** (5), 184–188.

Foley, J. and McMath, C. (1986). Dynamic Process Visualization, *IEEE Computer Graphics and Applications* **6** (3), 16–25.

Foley, J. D. (1990). Scientific Data Visualization Software: Trends and Directions. *International Journal of Supercomputer Applications* **4**, 154.

Grinstein, G, Pickett, R. and Williams, M. (1989). EXVIS: An Exploratory Visualization Environment, In *Proceedings Graphics Interface'89*, CIPS, Toronto, pp. 254–261.

Haber, R. (1989). "Scientific Visualization and the RIVERS Project at the National Center for Supercomputing Applications," *IEEE Computer* **22** (8), 84–89.

Haber, R. and McNabb, D. A. (1990). "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems." In B. Shriver and G. Nielson (eds), *Scientific Visualization*. IEEE Computer Society Press, Los Alamitos, CA.

Hai-Peng, C. and Landman, U. (1993). Controlled Deposition, Soft Landing, and Glass Formation in Nanocluster-Surface Collisions. *Science* **260** (May), 1304–1307.

Kim, W. and Foley, J. (1993). Providing High-level Control and Expert Assistance in User Interface Presentation Design. In *Proceedings INTERCHI 1993*, pp. 430–437. ACM, New York.

Kolodner, J. (1993). *Case-Based Reasoning*. Ablex.

Lefkowitz, H. (1991). Color Icons: Merging Color and Texture Perception for Integrated Visualization of Multiple Perception. In *Proceedings, Visualization '91*, San Diego, p. 164.

Mackinlay, J. (1986). Automating the Design of Graphical Presentations of Relational Information, *ACM Transactions on Graphics* **5** (2), 110–141.

Pickett, R. M. and Grinstein, G. G. (1988). Iconographic Displays for Visualizing Multidimensional Data. In *Proceedings of the 1988 IEEE Conference on Systems, Man, and Cybernetics*, Beijing and Shenyang, vol. 1, pp. 514–519.

Ribarsky, M. W., Hodges, L. F., Minsk, R. and Bruchez, M. (1993a). Visual Representations of Complex, Discrete Multivariate Data. *The Visual Computer*, accepted for publication.

Ribarsky, M. W., Ayers, E., Eble, J. and Mukherjea, S. (1993b). Using Glyphmaker to Create Customized Visualizations of Complex Data. Report GIT-GVU-93-25, 1993. (Submitted *IEEE Computer*.)

Ribarsky, W., Mukherjea, S. and Ayers, E. (1993c). Glyphmaker: An Interactive, Programmerless Approach for Customizing Visual Data Representations. Report GIT-GVU-93-26.

Roth, S. and Mattis, J. (1990). Data Characterization for Intelligent Graphics Presentations. In *Proceedings CHI '90*, April. pp. 193–200. ACM, New York.

Senay, H. and Ignatius, E. (1990). Rules and Principles of Scientific Data Visualization. Technical Report GWU-IIST-90-13, The George Washington University, Washington, DC, May. (Also in *State of the Art in Scientific Visualization*, SIGGRAPH '90 tutorial notes, August.)

Senay, H. and Ignatius, E. (1991). Compositional Analysis and Synthesis of Scientific Visualization Techniques, In Patrikalakis, N. (ed.) *Scientific Visualization of Physical Phenomenon*, Springer-Verlag, June, pp. 269–281.

Senay, H. and Ignatius, E. (1992). VISTA: A Knowledge Based System for Scientific Data Visualization, George Washington University Technical Report GWU-IIST-92-10, March. *IEEE Computer Graphics and Applications*, to appear.

Smarr, L. (1988). Supercomputing Offers New Challenges. *Supercomputing Review*, p. 39.

Smith, S., Grinstein, G. and Bergeron, R. D. (1991). Interactive Data Exploration with a Supercomputer. In *Proceedings Visualization '91*, San Diego, p. 248.

Treinish, L. A. and Gough, M. L. (1987). A Software Package for the Data-Independent Storage of Multidimensional Data. *EOS Transactions of the American Geophysical Union* **68** (28), 633–635.

Treinish, L. A. and Goettsche, C. (1991). Correlative Visualization Techniques for Multidimensional Data. *IBM Journal of Research and Development*, **35** (1/2), 184–204.

Treinish, L. (1993). Inside Multidimensional Data. *Byte* **18** (4), 132–133.

Tufte, E. (1983). *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT.

Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R. and van Dam, A. (1989). The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications* **9** (4), 30–42.

Ware, C. and Beatty, J. (1988). Using Color Dimensions to Display Data Dimensions, *Human Factors*, **20** (2), 127–42.

About the author

James Foley is Professor of Computer Science and Director of the Graphics, Visualization, and Usability Center at Georgia Institute Technology. He was previously Professor and Chairman of Electrical Engineering and Computer Science at The George Washington University. He earned his Ph.D. at the University of Michigan. His research interests include user interfaces and interactive computer graphics; his work, which has been sponsored by DEC, NSF, NASA, Sun, the Software Productivity Consortium and Siemens, focuses on building UIDE, the User Interface Design Environment. He is co-author, with A. van Dam, of *Fundamentals of Interactive Computer Graphics*, and with van Dam, S. Feiner and J. Hughes, of the recently-published *Computer Graphics: Principles and Practice*. Foley is a Fellow of the IEEE, and a member of ACM, the Human Factors Society and Sigma Xi. He is editor-in-chief of *ACM Transactions on Graphics*, serves on several editorial boards and consults regularly for governmental and industrial organizations.

William Ribarsky is Senior Research Scientist, Manager of the Scientific Visualization Lab, and Associate Director for Service of the Graphics, Visualization, and Usability Center, all at Georgia Institute of Technology. Dr Ribarsky's background is in computational physics. His recent research has been in programmerless methods for constructing visual representations of multivariate data, the semiotics of visual representations and the development of visual interfaces. Most recently he has been studying user-defined virtual spaces for scientific visualization including investigating interfaces, graphical representations, modes of interaction and methods of analysis.