Painting multiple views of complex objects

John Alan McDonald
Werner Stuetzle
Department of Statistics, University of Washington
Andreas Buja
Statistics Research Group, Bellcore

July 1, 1990

Abstract

This paper reviews and illustrates a direct manipulation approach to visualization of complex objects called *painting multiple views*. We describe a programming model for direct manipulation in general and for painting in particular, based on simple constraints between entities in an the underlying scientific database and the components of displays used to examine the data. With this model, the original notion of "brushing scatterplots" is easily extended.

Keywords: direct manipulation interfaces, exploratory data analysis.

1 Introduction

Suppose you want to display something complicated, like a large program, an automobile and all its parts, or some multivariate statistical data. A natural first thought is to draw a picture that's equally complicated, such as printing the entire program on the screen in a really small font, rendering the automobile and its parts as transparent solids, or presenting the data as a tableau of Chernoff faces [6].

Attempts at such dense encoding seldom work well. It's usually more effective to construct a number of simple, easy to understand displays, each focused clearly on a particular aspect of the information you are trying to convey. However, you then risk losing track of the relationships between isolated pieces of information.

The purpose of *painting multiple views* is to help the user to integrate scattered information—by marking corresponding parts of multiple displays with color (or some other form of highlighting).

For example, consider some experimental tools for browsing CLOS [25] programs, shown in figure 1. The windows are displaying a small part of Cactus, a system for numerical linear algebra and optimization [16]. The original motivation for constructing these displays was to understand some unexpected behavior observed when composing trivial `Identity` mappings with others. The editor, on the left, displays a few methods for the generic function `compose`. The class graph, at the upper right, focuses on the inheritance relations between a small subset of the `Mapping` classes; it makes the overall structure easy to grasp in a way that is not possible scrolling through definitions in the editor. At the lower right is an attempt at joint display of both class inheritance and methods for the generic function `compose`.

The exact nature of the problem with `Identity` is not relevant here. The point is that we are interested in the class `Identity`, there are references to it in all three windows, but, to find all the references, we have to *read* the displays carefully—their locations are not obvious from a quick glance at the screen.

It would help to have an easy way to highlight all references to `Identity`, wherever they appear on the screen. One way to do this is to allow the user to dip a mouse-controlled paint brush into a pallette of highlighting paint, which can be applied to the class `Identity` by moving the brush over any visible reference to that class. The result, using a highlighting paint corresponding to a large italic font, is simulated in figure 2. (The figures in this paper were produced on a Symbolics Lisp machine. Figure 2 is simulated because we have not implemented an editor; we are faking the result using the Zmacs editor [28].)

We generally find painting with color more effective. Here we are restricted to black and white illustrations, so we will highlight by change of font and/or size, which is less dramatic, but still useful.

Linking multiple views by highlighting or painting is a straightforward, if not obvious, application of direct manipulation. From our perspective, the key properties

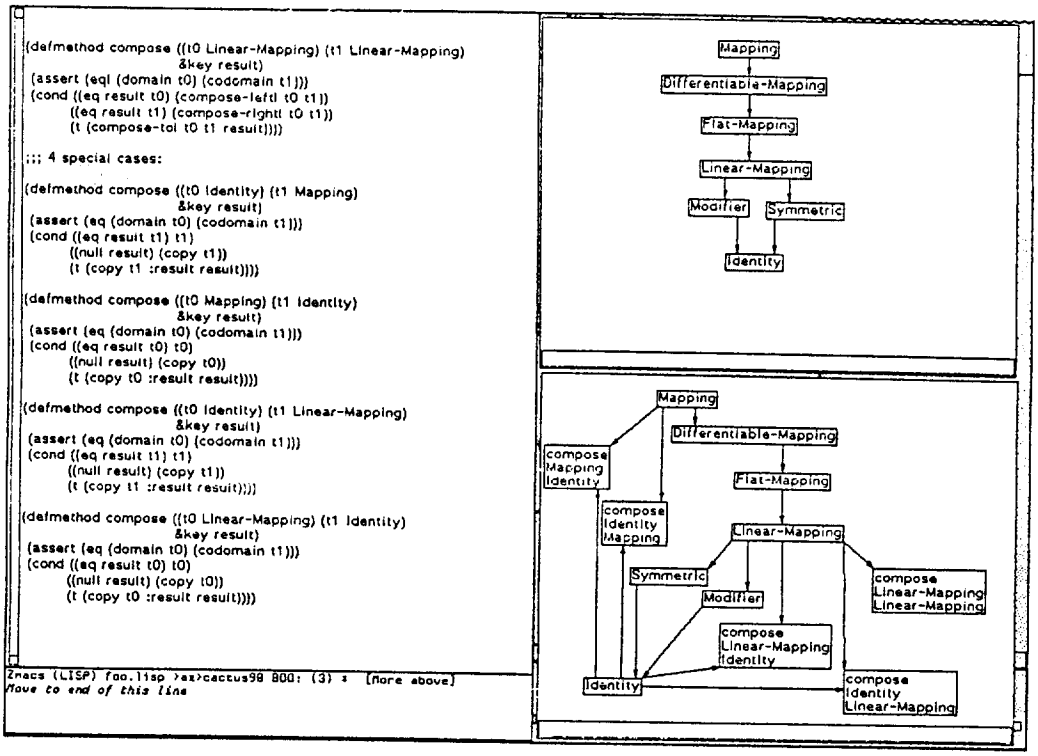

Figure 1: Experimental CLOS browsing tools.

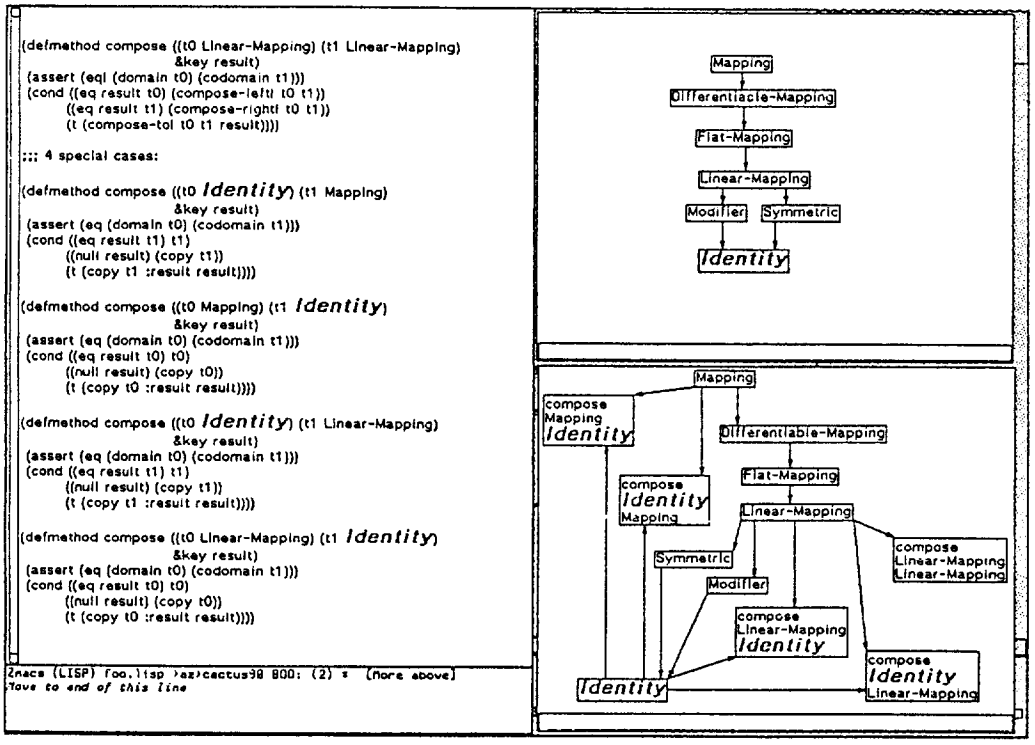

Figure 2: Simulation of painting the class `Identity`.

of a direct manipulation interface are:

- Displays are built out of visible (graphical or text) *presentations* of *subjects*. The subjects are entities in some underlying program or database.
- The appearance of a presentation changes automatically to reflect changes in the state of its subject.
- The presentation serves as a locus for graphical input that causes changes in the state of its subject.

To implement painting using direct manipulation, we assume that the state of subject includes or can be annotated with parameters that determine the color, font, size, etc., of its presentations. Mouse input to some presentation changes these display parameters in the subject, which causes all presentations to be re-drawn with the new color, font, size, etc.

Whether it's thought of as direct manipulation or not, this kind of interface architecture is common in object-oriented programming environments. Examples of systems that have or can easily support this structure are: LOOPS (Active Values) [4], Collab [26], Pogo [29], Garnet [18], Smalltalk (Model View Controller) [22], Thinglab [11], Animus [10], and Unidraw [31], among many others.

2 Brushing Scatterplots

One of the simplest applications that benefits from painting multiple views is statistical multivariate data analysis. In fact, our notion of "painting multiple views" arose as a generalization of an idea from interactive data analysis graphics, sometimes called "brushing scatterplots," which dates to the late 70's [19,15,3,2].

The statistician's multivariate data set is essentially a relation with n records and p Attributes, where each Attribute takes on floating point values. More geometrically, we can then think of it as a collection of n points—a *point cloud*—in a p dimensional euclidean space.

A fundamental problem in data analysis is to provide tools for seeing and understanding the p dimensional shape of such point clouds, without relying too heavily on preconceived notions of what might be going on in the data. This is intrinsically difficult, because human perception and imagination are designed for a world of one time dimension and 3 spatial dimensions.

The basic tool for looking at point clouds is the scatterplot, which can be thought of geometrically as a projection of the p dimensional points onto a two dimensional plane. There is a long history of attempts to encode more information in scatterplots. One alternative is encoding additional Attributes in the glyph or plotting symbol used for each data point. Another is to use realtime motion to encode one or more additional variables. An example is using rotation to display three dimensional point clouds. See [8] for a review and some recent examples of both these approaches.

"Brushing scatterplots" displays high dimensional point clouds by linking a number of simultaneous scatterplots. The linking is constructed interactively and dynamically modified by painting points in one scatterplot; all corresponding points in other scatterplots automatically take on the same color. This idea has become a standard part of many commercially available statistical graphics systems (e.g., MacSpin [9], Data Desk [30]).

To provide a better sense of how painting multiple views can be used, we next consider an example from the analysis of multispectral images. Figure 3 shows a Landsat image of the confluence of the Rio Solimões and the Rio Negro near Manaus, Brazil, taken by Landsat 2 on July 31, 1977. The full image consists of approximately 1000x1000 pixels, with intensities measured in four spectral bands for each pixel. Thus the image can be considered a 6 dimensional point cloud, with a point corresponding to each pixel, 4 dimensions corresponding to the 4 spectral bands and 2 dimensions to the x and y location of the pixel in the image.

Figure 3 was produced by the WISP image processing system [24]. WISP supports analysis of multispectral images using mixing models [1,23]. One purpose of mixing models is to infer what's on the ground from the spectral data recorded by the satellite. A mixing model decomposes the observed spectrum of a pixel (which, in Landsat MSS, covers approximately an 80 meter square) into a combination of a small number of spectra of "pure" substances. The idea is to estimate what proportions of the 80 meter square are covered by substances of interest (eg. rain forest vs. grassland). The issue addressed by this example is whether a simple (convex) linear combination of spectra is sufficient or a more complex, non-linear mixing model is required.

WISP has an interface that allows a subset of the pixel point cloud to be selected for analysis with the Data Viewer [5,13], a system for analyzing multivariate data using a variety of techniques based on realtime motion and interaction, including painting. One would prefer to analyze the whole image's point cloud, but the Data Viewer (and its hardware platform, a Symbolics 36xx series workstation) gives adequate interactive response with data sets of up to about 2 thousand points.

The image in figure 3 is overlaid with 5 small white rectangles (which may be difficult to see in black and white xerographic reproduction), outlining the pixels selected for examination with the Data Viewer: the small square in the upper part of the image covers pixels that are forest vegetation; the square in the lower right is grasslands; the square in the lower left is muddy water in the Rio Solimões; the square in the middle left is dark water in the Rio Negro; and the long narrow rectangle in the middle is a transect across the confluence (mixing) of the muddy and dark water.

The selected pixels are displayed in two Data Viewer windows, shown in figure 4: The left window, with its label `Brushing` obscured by the menu that begins `Highlight all`, shows a scatterplot of the original x and y pixel coordinates. The viewing transformation for this window has been chosen so that only the muddy water, dark water, and transect pixels are visible. The right plot, labeled `Variable Plots`, shows a scatterplot of Band-5 vs Band-6 for all 5 groups of pixels. (Landsat was originally intended to have 7 spectral bands; due to a malfunction, the satellite only returns bands 4 through 7.) The key question to be answered is whether the pixels in the transect can be represented as a convex combination of muddy and dark water spectra. This is true if the transect pixels lie along a line between the muddy and dark water pixels in the four dimensional spectral space.

In figure 4, muddy water pixels, in the left bottom of the Brushing window, are highlighted, by covering them with the paint brush (the small rectangle). Unhighlighted pixels are drawn as horizontal dashes and highlighted pixels are drawn as vertical dashes. The muddy water pixels fall at the top of the linear structure on the left of the points in the `Variable Plots` window.

In 5, dark water pixels, at the far left of the `Brushing` window, are highlighted. The dark water pixels turn out to be at the bottom of the linear structure.

Figures 6 and 7 show a sweep across the transect of the confluence, which produces a corresponding sweep across the linear structure in the `Variable Plots` window, confirming that these pixels are in fact, convex combinations of dark and muddy water pixels.

3 Implementation

The need to solve concrete scientific and engineering problems is a tremendous stimulant of creativity, especially in ways of displaying and interacting with specialized kinds of information. It follows that an ideal system for visualization should help users to improvise new kinds of displays quickly, including new modes of interaction with those displays. Failing that, a good system should permit experts to implement small extensions of existing functionality quickly. For this to be true, it must be possible to implement a change that seems small to the user by a small change to the program. This implies that the abstractions used in the program must correspond closely to the way the users think about the underlying problem.

It may seem that we are belaboring an obvious point, but it is rare in scientific computing to find systems whose programming abstractions are at all similar to the underlying problem they are intended to solve.

Consider, for example, what we will call the "standard" implementation model for brushing scatterplots. In the standard implementation, each scatterplot has an $N \times 2$ array holding the X and Y coordinates of the points. This array is created typically by extracting two-dimensional records from an underlying database and applying a viewing transformation to get window coordinates. In addition, there is a color array of length N that is shared by all plots. When brushing, the program examines the XY array to find the indices of the points under the mouse, changes, if necessary, the values in the corresponding locations in the color array, and redraws points at the coordinates in the corresponding locations in all the XY array.

This implementation strategy is natural for a programmer with experience in conventional scientific computing; Fortran subroutine libraries often represent important information purely by location in arrays. Nearly all implementations of brushing scatterplots of which we are aware, including our own early work, share the essential feature (and failing) of this model, which is that the correspondence between plots is represented by matching array indices. Unfortunately, it precludes even the most trivial extensions. For example, suppose the user wishes to simultaneously analyze three data sets: all the patients in a clinical trial of some drug, the male patients in the trial, and the female patients. Typical commercial implementations of brushing scatterplots either restrict the user to a single data set, or brush separate data sets separately, ie., brushing in a plot of all patients has no effect on plots of male patients or plots of female patients.

If we think of brushing scatterplots in terms of direct manipulation, then it's obvious what wrong with the "standard" implementation: There's no reliable representation for the identity of the underlying entity that's being brushed.

We next present an implementation model for painting multiple views based on direction manipulation. The following discussion is based primarily on the Antelope system [14,17] and to a lesser degree on the Plot Windows [27] and Data Viewer [5,13] systems. The model presented here is not an accurate description of any of these systems; it reflects what we have learned from what was wrong with them. It is the basis for a new, more portable system, currently under development.

To successfully support direct manipulation, our programming model has several components that are implicit in a conceptual model of direct manipulation (see Figure 8):

- Subjects in a Database

  There is an object (a *Subject*) in the programming environment representing each abstract entity (eg., a multispectral image pixel) that the user might want to examine, select with the mouse, and ma-

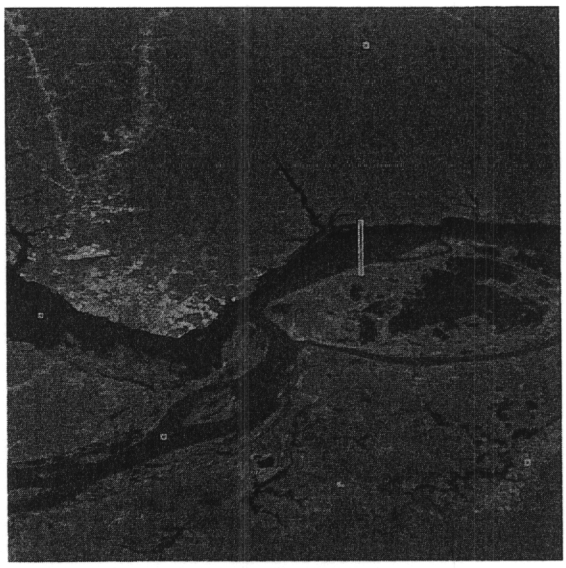

Figure 3: The confluence of the Rio Solimões and the Rio Negro.

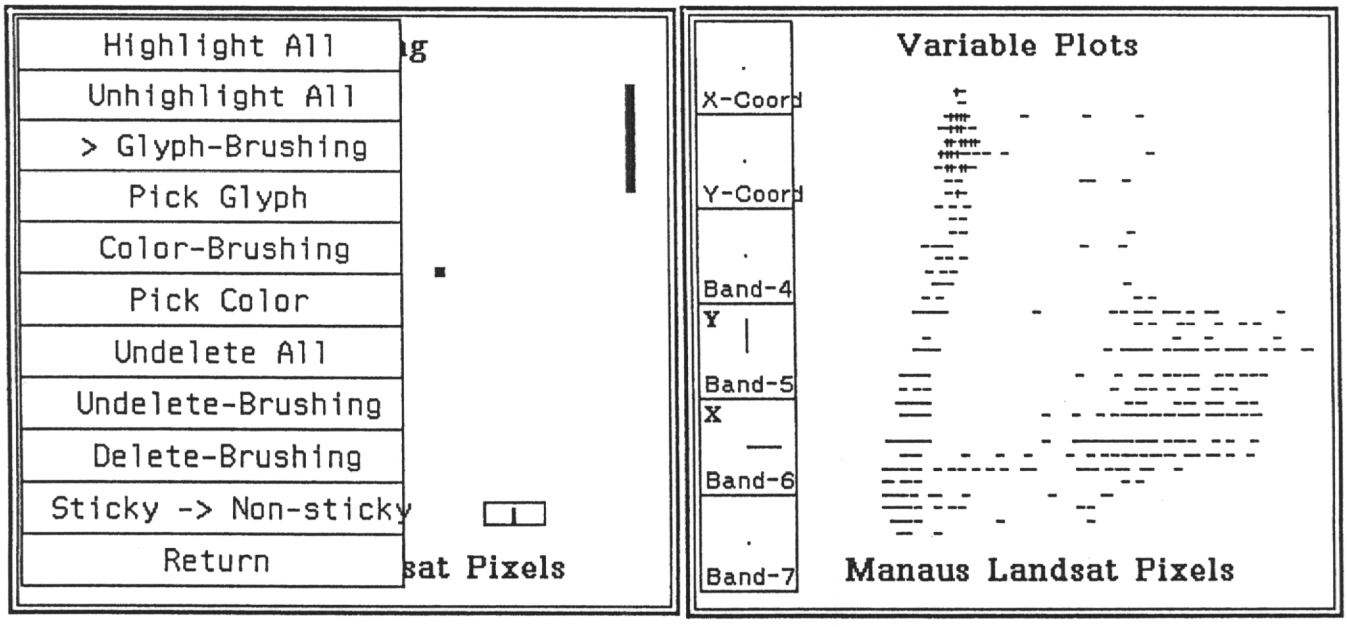

Figure 4: Highlighting muddy water.

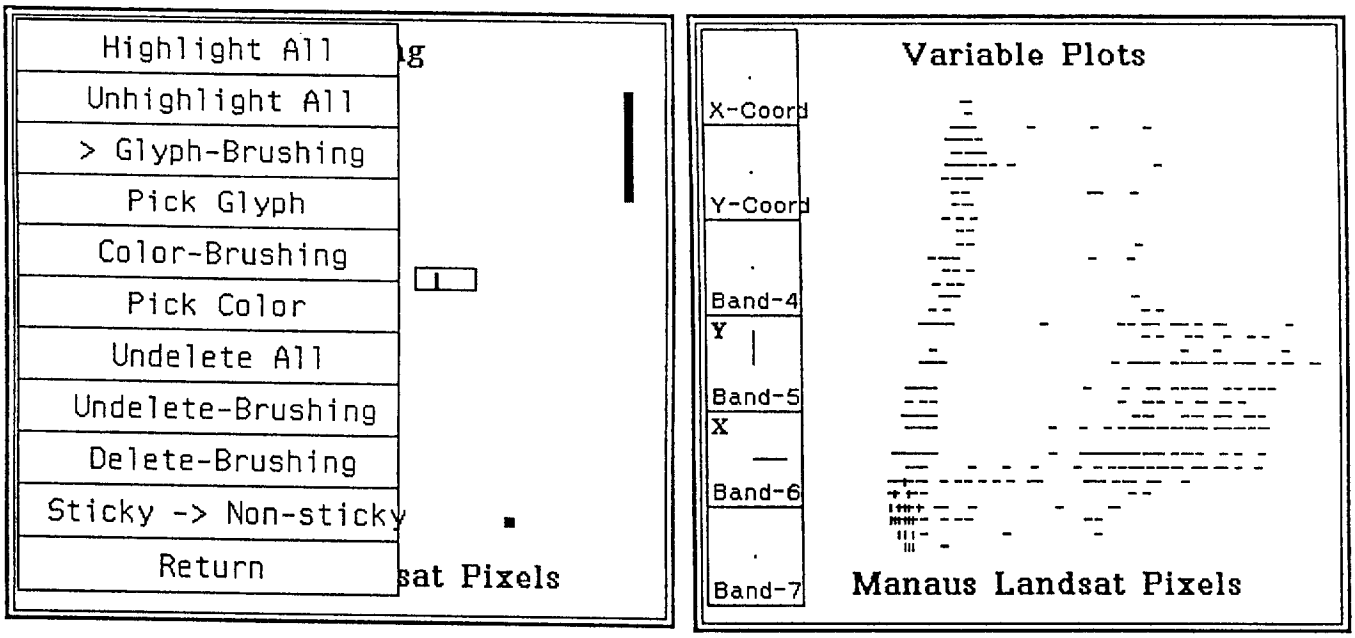

Figure 5: Highlighting dark water.

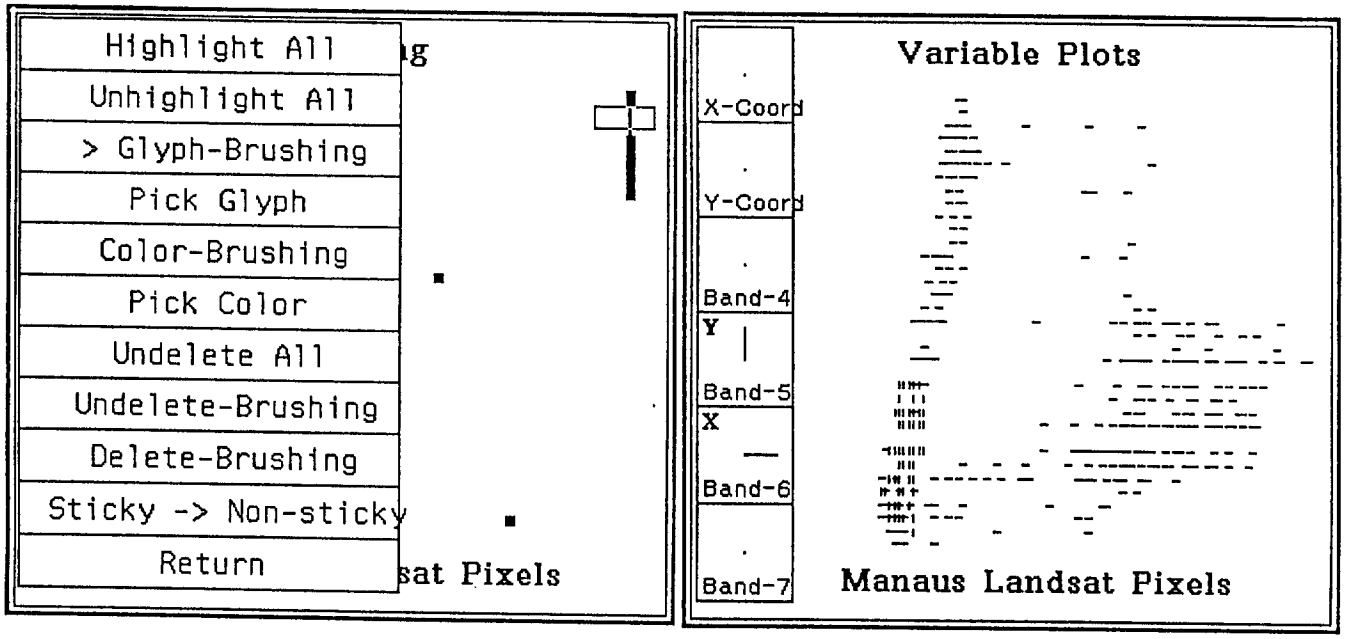

Figure 6: Highlighting the top of the confluence.

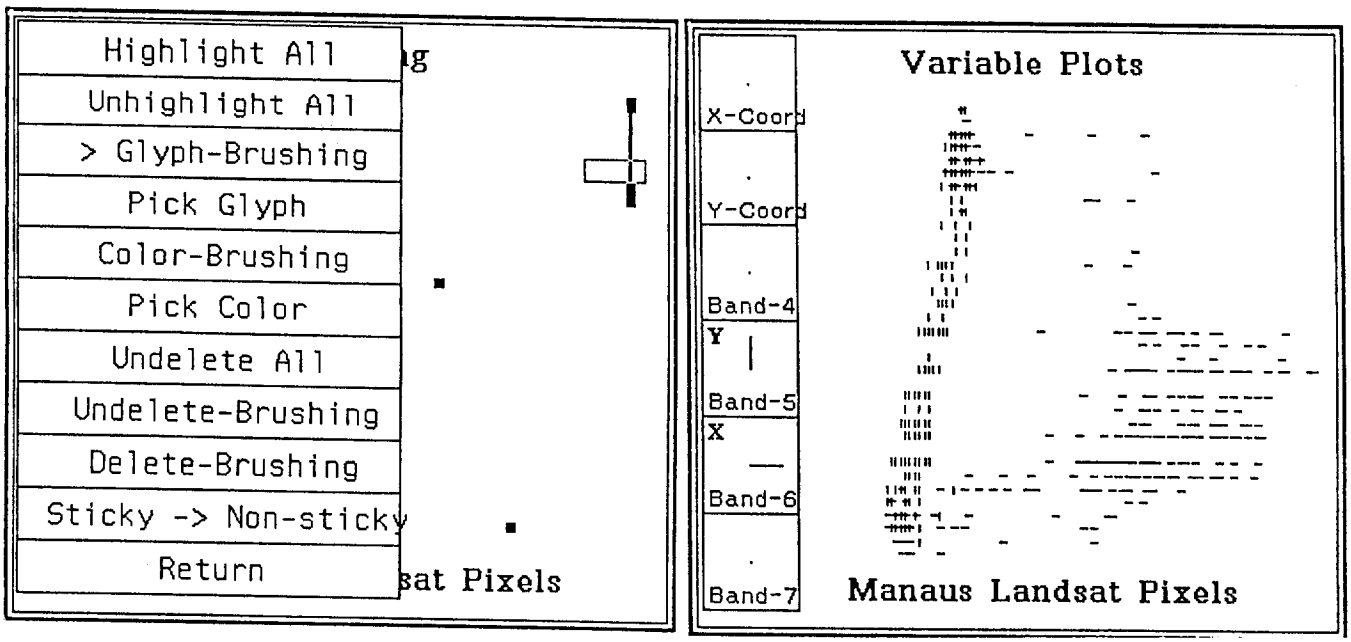

Figure 7: Highlighting the whole confluence.

nipulate.

Each Subject has associated with it a set of *Attributes*; an Attribute has an identity independent of any particular Subject. We can evaluate an Attribute for any Subject for which it is defined and we can change the value for some, if not all, Attributes. We can also define new Attributes for a given Subject at runtime. This is similar to the data model proposed in [21].

Mutable Attributes are required to support the mutable appearances of the Subject's Presentations. Having an extensible set of Attributes allows different Presentations to behave differently in ways decided at runtime (see section 4.2 below).

In some object-oriented programming languages, it is straightforward to represent Subjects by objects and Attributes by the slots (instance variables) of the Subject objects. Unfortunately, in many languages and databases, it may too expensive or not even possible to have mutable slots or to add slots at runtime. However, we can use essentially the same programming model as long as Subjects have unique identities and we can build mutable tables that map Subject identities to colors or other appearance parameters. We can then generalize our notion of Attribute to include both slots that are intrinsically part of each Subject and these mutable tables, defined at runtime.

The key points are that we can: (1) get values for any defined Subject-Attribute pair (2) set values for all Subjects and some Attributes and (3) define new mutable Subject-Attribute pairs at runtime.

Our programming model also presumes that Subjects can be organized into structured Collections, like the Collection classes of Smalltalk [12], eg. lists, directed graphs, or dictionaries. We also presume that new types of Collections can be defined — if not by a random user, then at least by a programmer sophisticated enough to develop new kinds of plots.

- Presentations in a Scene

  The *Scene* consists of a directed acyclic graph (usually a tree) of objects (Scene Nodes) representing geometric shapes (including text), in a two or three dimensional Scene Space. The Scene is rendered onto a display surface by standard computer graphics techniques. Some nodes in the Scene are visible representations or *Presentations* of some of the Subjects. (Our use of the word "Presentation" is similar, but not identical to [7].)

  A Subject determines the position and appearance of its Presentations through an output Lens (see below). The hierarchical or graph structure of the Scene will often reflect a similar hierarchy in the Subjects; in other words, a Presentation of a Collection may have children that are Presentations of the elements of the Collection.

- Links

  There must be a mechanism for linking Presentations and their Subjects. There are three kinds of connections between Presentations and Subjects:

  - Lenses for output filtering

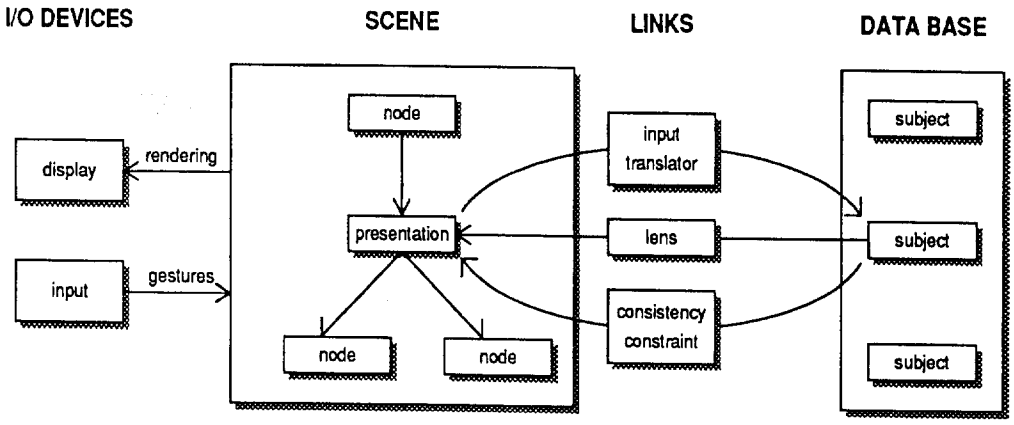

Figure 8: A diagram of the programming model.

A *Lens* maps from Subjects to be visualized into draw-able colored geometric shapes (including text) in the Scene space; it computes the position and appearance of a Presentation from the Attributes of its Subject.

In general, each Presentation could have its own unique Lens, computing position, shape, size, color, texture, etc., from its Subject however it likes. However, the standard painting operation—and most of the interesting variations—can be implemented with a small number of simple Lenses, shared across Presentations and across plots as well.

- Input-Translators

  An *Input-Translator* maps input events received by a Presentation to operations on its Subject.

- Consistency-Constraints

  A *Consistency-Constraint* ensures that the Scenes always reflect the current state of the Subjects.

To implement a simple version of painting for scatterplots, we first suppose the Subjects (the records in the multivariate dataset) have p quantitative Attributes and one color Attribute. We also suppose the records are organized into Data-Set collections; a Data-Set is essentially a list of records, with perhaps additional (summary) information about its records and Attributes.

In a scatterplot, the Scene is a tree. A Scatterplot-Root node typically has four children: a Label node that is a Presentation of a Data-Set, X-Axis and Y-Axis nodes, and a Point-Cloud node (another Presentation of the Data-Set). An Axis node has a Label node (a Presentation of a continuous Attribute) and a Ruler node as children; a Ruler has Tic and possibly Tic-Label children. A Point-Cloud has Point-Glyph children, each of which is a Presentation of an element of the Data-Set.

A scatterplot uses a simple scatterplot Lens with two component sub-Lenses: (1) a Projection-Lens that maps a high dimensional record to a position in 2D Scene space and (2) a Color-Lens that reads the value in the color slot in the record object. The Projection-Lens varies from plot to plot, but is the same for all Point-Glyph Presentations in a given plot. All the Point-Glyphs in all plots share the same Color-Lens, so that all Point-Glyphs always have the same color as their Subjects.

Painting input events received by a Point-Glyph are translated to the operation of changing the color Attribute of the glyph's Subject record. The Consistency-Constraint's ensure that all visible Point-Glyph's are redrawn when their Subject's color Attribute changes.

4 Small extensions of brushing scatterplots

In this section, we present a number of small variations on the basic idea of "brushing scatterplots." in the next section we discuss the slightly larger extension to displays of discrete data. There are several reasons for doing so. One is to present a small sample of the range of applications of painting multiple views, all of which are natural consequences of the user's thinking in terms of direct manipulation of some underlying database. Another is to give some sense of how rapidly small new ideas can arise, given the stimulation of new problems and a simple but powerful conceptual model. This is more evidence of the importance of improvisation in visualization systems. Finally, we wish to demonstrate that our simple programming model easily supports implementation of these few explicit examples and, by implication, many other small extensions as well.

4.1 Connecting points by lines

Our programming model supports a number of variations of painting using Collections that have more structure than simple lists of record objects.

The basic idea of linking multiple scatterplots by changing the appearance of points was first invented by Newton in 1978 [19]. Newton's system provided an interesting variation on painting: a user could connect pairs of points by lines; the lines would be drawn in all plots.

This operation makes most sense if we consider our plots to be views of directed graphs, whose nodes are records. Instead of multiple scatterplots, we have multiple graph editors. The operation of adding and deleting edges in a graph may overstretch the "painting" metaphor somewhat, but it is fully consistent with "direct manipulation": Adding an edge is a modification of an object in the underlying database (the graph), and is therefore reflected automatically in all views of that object.

4.2 Painting Rings

Certain variations of painting can be easily implemented using simple, but differing Color-Lenses in different plots.

For example, some painting systems require plots to be explicitly linked, permitting the definition of disjoint *painting rings*; painting can be done between plots within a ring, but not across rings. Our model supports this by having multiple color Attributes in the subjects, one for each plot ring, and a distinct Color-Lens for each ring.

4.3 Shadow Highlighting

Becker and Cleveland's *shadow highlight* [3,2]. operation offers another variation. In shadow highlight mode, all records are drawn as either highlighted or lowlighted in the active plot (the one containing the brush). In other plots, only the records corresponding to the highlighted points in the active plot are drawn; records corresponding to lowlighted points are not drawn (or are drawn with invisible paint).

Shadow highlighting can be implemented by using the standard Lens in the plot containing the mouse and a Lens that translates `lowlighted` to `invisible` in all the other plots.

4.4 Markup and commit

One example of the use of the Input Translator is to allow temporary violation of the direct manipulation model, which assumes that all redrawing of windows is done to reflect changes in the underlying database. Delayed satisfaction is a violation of the conceptual model, but it is sometimes necessary to get fast-enough feedback. This is important if, for example, one implements painting on images without subsetting. It can be thought of as a variation on the idea of markup, that is, the user gets a chance to see what the result will look like before committing to a change in the database.

Suppose one Input Translator is shared by all the Presentations in a plot. It can collect and save painting input events, and cause the Presentations that receive them to be redrawn, without notifying or modifying the Subjects in the underlying database and without triggering the possibly expensive consistency constraint mechanism. When a `commit` event is received, the painting operations can be forwarded to the database, possibly re-packaged in a more efficient form (e.g., multiple re-painting of the same Subject can eliminated). If a `cancel` event is received, the Translator can discard the saved events and cause the Presentations to be redrawn with their original appearance (possibly by triggering the constraint mechanism).

5 Painting tables of discrete data

Scatterplots are the display of choice for continuous Attributes, like height, weight, or blood pressure. Statisticians, however, often encounter data with *categorical* Attributes, that is, Attributes that take on a small number of discrete values, like {male, female} or {Cadillac, Ford, BMW, ...}.

Our next example concerns some categorical data obtained from Mieko Ogura and Bill Wang of the Project on Linguistic Analysis at UC Berkeley [20]. Early in this century, linguists recorded, in each of 311 villages in England, the principal vowel sound (or *reflex*) used in the pronunciation of each of 181 words. Altogether, there are 113 possible vowel reflexes. However, the data is somewhat simplified by being grouped into families, like the 35 `EA` words (eg. `great`, `beans`). Within each family, only a subset of the possible reflexes occurs. For example, only 20 distinct reflexes are used for `EA`.

Taking just the 35 `EA` words, we could represent this data by a collection of 311 village record objects, with 37 Attributes—2 continuous Attributes corresponding to latitude and longitude and 35 categorical Attributes, one for each of the `EA` words. A scatterplot of latitude vs. longitude for the villages is shown in the lower right of figure 9 (among a collection of windows produced by the Antelope system [14]); the points fill out the outline of England.

Categorical data is often summarized by contingency tables. A one-way, or, better, one dimensional contingency table merely counts the number of records that have each possible value of a given categorical Attribute. A two dimensional contingency table counts the number

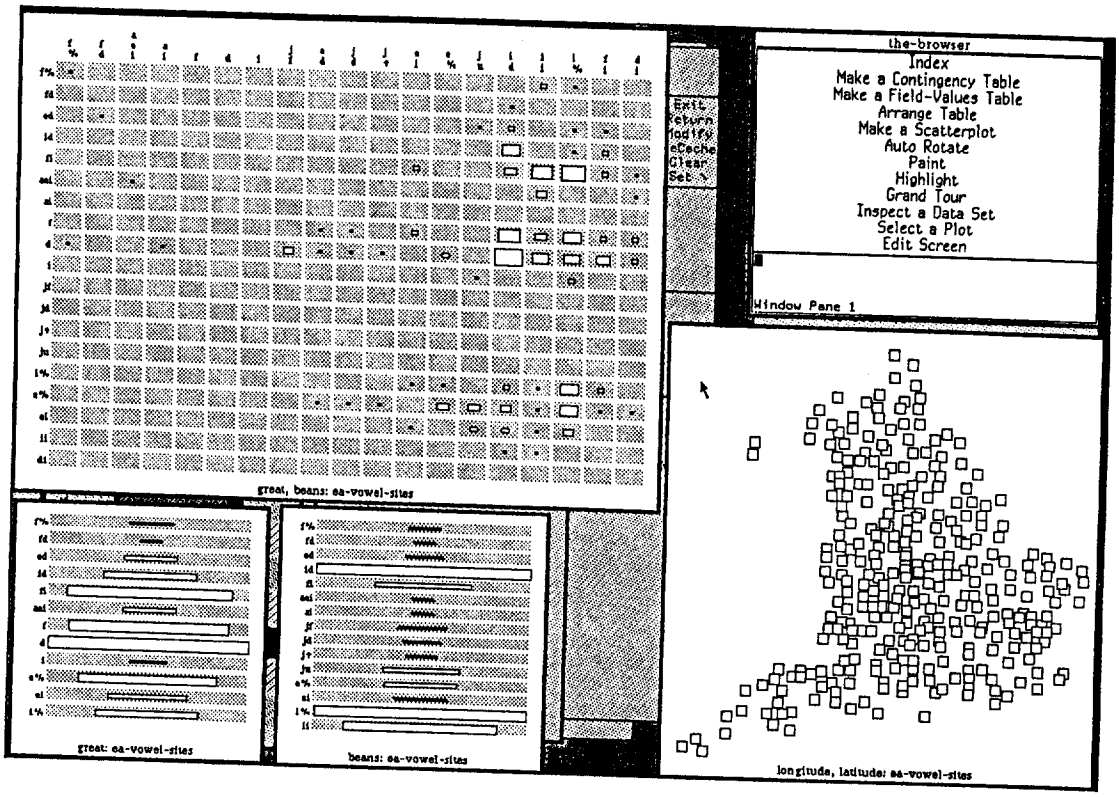

Figure 9: English Vowel data.

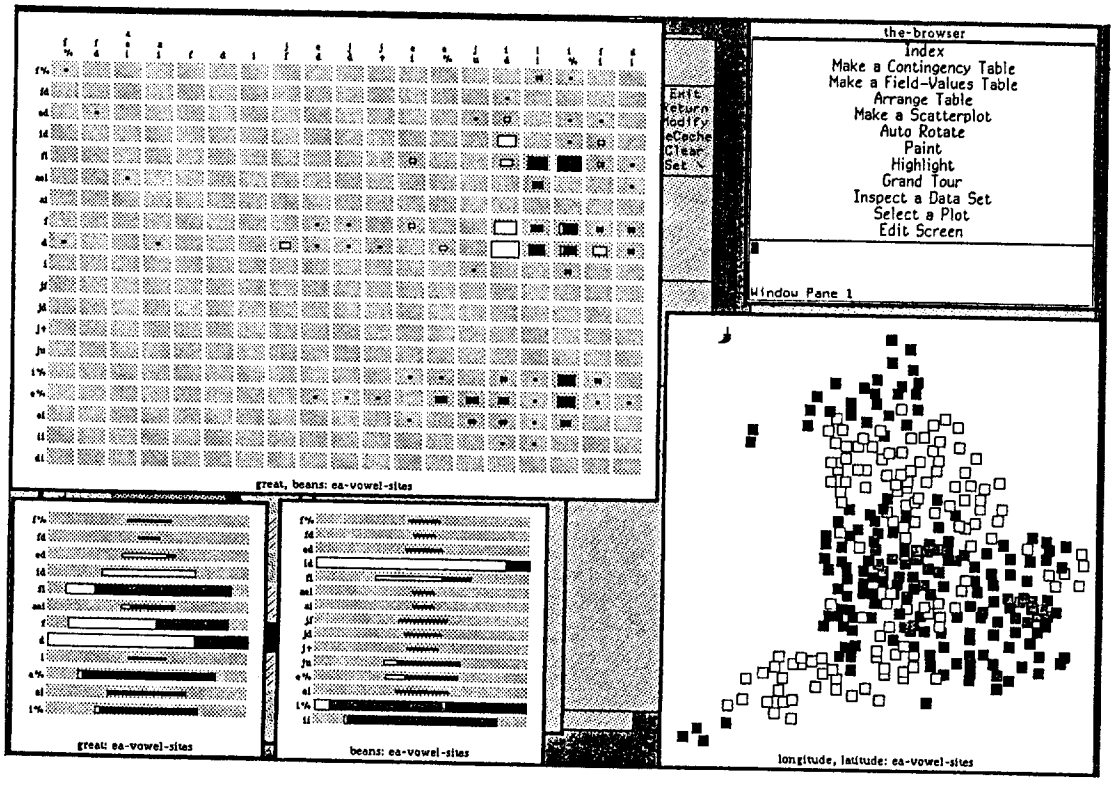

Figure 10: English Vowel data after painting.

of records that have each possible pair of values for a given pair of Attributes.

Figure 9 shows graphical representations of 3 contingency tables.

In the bottom left corner, is a one dimensional contingency table for the word **great**. There is one cell in the table for each of the twelve reflexes that are used to pronounce **great** somewhere in England. The area of the white rectangle is proportional to the number of villages that use that reflex; the most common reflex is the one that's labeled **d**. (The reflex labeling is a fairly arbitrary translation of phonetic symbols into one or two ASCII characters.)

Just to the right of the **great** table is a similar table for **beans**. Fifteen different reflexes are used to pronounce **beans**; the two most common are labeled **id** and **i%**.

In the upper right corner is a two dimensional contingency table, with the value of **great** plotted on the horizontal and the value of **beans** plotted on the vertical. Here, the area of the white rectangle is proportional to the number of villages that use a particular pair of pronunciations for **great** and **beans**. The most common pair is labeled **d** for **great** and **id** for **beans**; the (**fi**,**i%**) pair is almost as common.

An obvious question is how pronunciation is related to spatial distribution in the latitude-longitude scatterplot. This can be answered easily by painting both points in the scatterplot and cells in the contingency tables. The result is figure 10, which shows spatial coherence in pronunciation, ie., regional dialects.

Painting a point in a scatterplot changes the color of one record. Painting a cell in a table is equivalent to changing the color Attribute of all the records that fall in that cell.

A point in a scatterplot is a presentation of a single record. A cell in a table is a presentation of a set of records, which are usually not all the same color. Cells display the colors of their records using a divided color bar; the proportions of the rectangle that are black and white are the same as the proportions of black and white records in the set that the cell represents.

6 Conclusion

There is a natural tendency to attempt to convey complex information with displays of equivalent complexity—in other words, to encode as much as possible in a single picture. However, such dense encoding is not always possible. When it is possible, it may not be the most effective way to present what's important. Indeed, there is often more than one way to present the same information; which way is preferred depends on the task to be accomplished.

Thus, it is often better (or, at least, easier) to present complex information in a number of simple, separate, easy-to-comprehend pictures, each focused on a particular aspect of the larger information system.

Linking multiple views by painting is an effective method for visualizing complex information. Painting is best thought of as direct manipulation of the underlying entities. This paradigm, if suitably refined, leads to a programming model whose components consist of (1) manipulated Subjects in a database, (2) graphical Presentations of Subjects, and (3) Links which (a) map a Subject to the appearance of a Presentation, (b) translate input events into operations on Subjects, and (c) update Presentations in response to changes in the database.

7 Acknowledgments

The research was supported by Bell Communications Research, the Dept. of Energy under contract FG0685-ER25006; the Office of Naval Research under Young Investigator award N00014-86-K-0069, contract N00014-83-K-0472, and grant N00014-84-G-0178; and the National Science Foundation under grant DMS-8504359. Part of this manuscript was completed while the second author was with Bellcore. A number of important ideas used here (in particular the notion of Lenses) came from discussions with Alan Borning. We thank John Michalak for producing figure 8 and Steve Willis for producing figure 3.

References

[1] John B. Adams, Milton O. Smith, and Alan R. Gillespie. Simple models for complex natural surfaces: A strategy for the hyperspectral era of remote sensing. In *IGARRS'89, the IEEE International Geoscience and Remote Sensing symposium*, 1989.

[2] R.A. Becker and W.S. Cleveland. Brushing scatterplots. *Technometrics*, 29:127–142, 1987.

[3] R.A. Becker and W.S. Cleveland. Dynamic graphics for data analysis. *Statistical Science*, 2:355–395, 1987.

[4] D.G. Bobrow and M. Stefik. *The LOOPS Manual.* Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, 1983.

[5] Andreas Buja, Catherine Hurley, and John Alan McDonald. A Data Viewer for multivariate data. In *Computer Science and Statistics: Proc. 18th Symp. on the Interface*, pages 171–174, Washington, D.C., 1987. ASA.

[6] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68:361–368, 1973.

[7] E.C. Ciccarelli. Presentation based user interfaces. Technical Report 794, MIT AI Laboratory, 1984.

[8] W.S. Cleveland and M.E. McGill. *Dynamic Graphics for Statistics.* Wadsworth and Brooks/Cole, Belmont, Ca., 1988.

[9] A.W. Donoho, D.L. Donoho, and M. Gasko. MacSpin: dynamic graphics on a desktop computer. *IEEE Computer Graphics and Applications*, 8(4):51–58, 1988.

[10] Robert Duisberg. *Constraint-based animation: the implementation of temporal constraints in the Animus system.* PhD thesis, U. of Washington, 1986. Available as Tech. Rept. 86-09-01, Department of Computer Science, U. of W., Seattle WA 98195.

[11] R. K. Ege, D. Maier, and Alan H. Borning. The Filter Browser—Defining Interfaces Graphically. In *Proceedings of the European Conference on Object-Oriented Programming*, Paris, 1987. Association Francaise pour la Cybernetique Economique et Technique.

[12] A. Goldberg and D. Robson. *Smalltalk-80, The Language and Its Implementation.* Addison-Wesley, Reading, MA, 1983b.

[13] Catherine Hurley. *The Data Viewer: a program for graphical data analysis.* PhD thesis, Dept. of Statistics, U. of Washington, 1987.

[14] John Alan McDonald. Antelope: data analysis with object-oriented programming and constraints. In *Proc. of the 1986 Joint Statistical Meetings, Stat. Comp. Sect.*, 1986. Also Tech Rept 89, Dept. of Statistics, U. of Washington.

[15] John Alan McDonald. Interactive graphics for data analysis. In W.S. Cleveland and M.E. McGill, editors, *Dynamic Graphics for Statistics*, pages 247–275. Wadsworth and Brooks/Cole, Belmont, CA, 1988. Ph.D. thesis, Dept. of Statistics, Stanford, June 1982, and Dept. of Statistics, Stanford, Tech Rept Orion 11.

[16] John Alan McDonald. Object-oriented programming for linear algebra. *SIGPLAN Notices (Proceedings OOPSLA'89)*, 24(10):175–184, 1989. also Tech. Rept. 175, Dept. of Statistics, GN-22, U. of Washington, Seattle, WA 98195.

[17] John Alan McDonald and Jan O. Pedersen. Computing environments for data analysis III: Programming environments. *SISSC*, 9(2):380–400, 1988.

[18] Brad A. Myers. An object-oriented, constraint-based user interface development environment for X in Common Lisp. In *4th Annual X Technical Conference: Technical Papers, Boston Marriot Copley Place, January 15-17, 1990*, 1990.

[19] C. M. Newton. Graphics: from alpha to omega in data analysis. In P.C.C. Wang, editor, *Graphical Representation of Multivariate Data.* Academic Press, New York, 1978. Proceedings of the Symp. on Graphical Representation of Multivariate Data, Naval Postgraduate School, Monterey, Ca., Feb 24, 1978.

[20] Mieko Ogura and William Wang. Spatial distribution of the Great Vowel Shift in England. In *Language Transmission and Change.* Blackwell, London, England, 1989.

[21] R.W. Oldford. Object-oriented software representations for statistical data. *Journal of Econometrics*, pages 227–246, 1988.

[22] Lewis J. Pinson and Richard S. Wiener. *An introduction to object-oriented programming and Smalltalk.* Addison-Wesley, Reading, MA, 1988.

[23] Antonio Possolo, John Adams, and Milton Smith. Mixture models for multispectral images. Technical report, Dept. of Geological Sciences, U. of Washington, 1989.

[24] P. Shippert, G. Bradshaw, and S. Willis. *Washington Image and Spectral Package (WISP) Preliminary Documentation.* Remote Sensing Laboratory, Dept. of Geological Sciences, U. of Washington, Seattle, WA 98195, May 11, 1989.

[25] G.L. Steele. *Common Lisp, The Language, 2nd Edition.* Digital Press, 1990.

[26] M. Stefik, G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *CACM*, 30(1):32–47, 1987.

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

[28] Symbolics. *Text Editing and Processing.* Symbolics, Inc., 4 New England Tech Center, 555 Virginia Road, Concord, Mass 01742, 1986.

[29] Mark A. Tarlton and P. Nong Tarlton. Pogo: a declarative representation system for graphics. In Won Kim and Frederick H. Lochovsky, editors, *Object-oriented concepts, databases, and applications*, chapter 7, pages 151–176. ACM Press, New York, 1989.

[30] Paul F. Velleman. *Learning Data Analysis with Data Desk*. W.H. Freeman and Co., New York, NY, 1989.

[31] John M. Vllisides and Mark A. Linton. Unidraw: A framework for building domain-specific graphical editors. In *4th Annual X Technical Conference: Technical Papers, Boston Marriot Copley Place, January 15-17, 1990*, 1990.