Plot Windows

WERNER STUETZLE*

Personal workstations with bitmap displays and window systems are becoming a widely used computing tool. This article describes some of the capabilities of an experimental package for drawing scatterplots and histograms that has been implemented on such a workstation. An example illustrates how those capabilities can be used in the graphical analysis of a data set.

KEY WORDS: Workstation; Scatterplot; Desk top; Object-oriented programming; Window system.

1. INTRODUCTION

Most data analysis in the future will be done on personal workstations. A typical workstation executes a million instructions per second, has a megabyte of memory, and a bitmap display with a resolution of 1,000 × 1,000 pixels. It is used by one person at a time. The screen usually can be divided into several windows assigned to different tasks. A graphical input device such as a mouse controls the location of a cursor on the screen and thus allows graphical input. Machines that fit the description are offered by a number of computer companies (see McDonald and Pedersen 1985a,b).

The goal of this article is to illustrate how the capabilities of a personal workstation can be used for better statistical graphics. It draws extensively on two areas of research. The first one is research on human interfaces and screen organization, done mostly at the Xerox Palo Alto Research Center (PARC). The ideas developed at PARC form the basis of the Macintosh user interface and have achieved considerable popularity. The second line of ancestry can be traced to the work of Fowlkes (1971) on interactive choice of data transformations, the pioneering PRIM-9 system (Fisherkeller, Friedman, and Tukey 1974), and later systems for graphical data analysis, such as PRIM-H (Donoho, Huber, Ramos, and Thoma 1982), ORION (Friedman, McDonald, and Stuetzle 1982), and MacSpin (Donoho, Donoho, and Gasko 1985). These systems have demonstrated the use of dynamic, interactive graphics for the visual inspection and interpretation of multivariate data.

Plot Windows is a package of functions for drawing scatterplots and histograms that has been implemented on a Symbolics 3600 Lisp workstation. This article is intended to describe some of its functionality. Implementation techniques and design issues were discussed in Stuetzle (1986). Reasons for the choice of a Lisp environment as a basis for the implementation can be found in McDonald and Pedersen (in press). All figures are snapshots of the Symbolics screen.

Systems with capabilities similar to those of *Plot Windows* will undoubtedly be written for other workstations. The *Data Desk* package for the Macintosh (Velleman 1986) represents a step in this direction.

2. THE DESKTOP CONCEPT

Most statistical languages now in use, such as Minitab (Ryan, Joiner, and Ryan 1976), ISP (PC-ISP 1985) and S (Becker and Chambers 1984), were originally designed for keyboard interaction from terminals connected to a computer through relatively slow (10–1,000 characters per second) serial lines. The user types commands; the system scrolls the screen and displays its response. This means that text and plots, once displayed, disappear as more commands are entered and answered. Unlike data, functions, or macros, plots are not persistent objects. The only way to save a plot is by making a hardcopy. Typically, plots also support little or no interaction—they are merely images that can be looked at.

On a personal workstation we can do better than that. The screen is divided into a number of rectangular *windows,* as illustrated in Figure 1. The large *Editor* window in white on the lower left is the window at which we type commands to have them executed. Commands, in our case, are simply statements in the Lisp language, and the *Editor* window comes as part of the Symbolics Lisp environment. Whatever we type is stored in the editor buffer, which can be saved if we wish to keep a record of the commands issued during a session. We can scroll the buffer, cut and paste, go back to commands typed some time ago and select them for evaluation, and so forth. The ability to execute commands out of an editor is a major convenience. It also makes it very easy to package frequently used command sequences as new functions that can be invoked by a single command.

Commands produce two different kinds of results. First, every command is a Lisp expression and upon evaluation returns a value, which can be a number, an array, or a more complex object, such as a scatterplot. This value or, more precisely, a printed representation thereof is automatically displayed in the *typeout pane* at the bottom of the *Editor* window. The typeout pane scrolls like a conventional terminal screen. Second, commands can have side effects. A command to draw a scatterplot not only returns the scatterplot as its value; it also creates a new window on the screen, in which the plot appears. Similarly, a command to perform a regression analysis might result in the creation of several new windows, displaying the regression coefficients, residual plots, and so forth. In any

* Werner Stuetzle is Associate Professor, Department of Statistics, University of Washington, Seattle, WA 98195. The author wishes to thank Andreas Buja, Catherine Hurley, and John McDonald for their careful reading and criticism of an earlier draft of this article. Special thanks go to John McDonald for many valuable suggestions and generous help in programming and debugging. This research was supported by National Science Foundation Grant DMS-8504359 and U.S. Department of Energy Grant DE-FG06-85-ER25006.

case, results do not appear interspersed with the commands themselves.

Apart from the command window, Figure 1 shows three more exposed windows, a scatterplot (*Plot1*) at the top right, the *Data-Inspector* in the middle right, and a folder (*All-Windows*) at the bottom right. Those types of windows will be discussed further. Other windows are partially visible in the gray area of the screen. Let us ignore those.

We can think of this whole arrangement as a desk top. Each window and, in particular, each plot corresponds to a sheet of paper on the desk top. Generating a new plot places a new sheet of paper on the desk.

3. SHUFFLING PAPER ON THE DESK

We can perform the same operations on windows—virtual sheets of paper—on a virtual desk that we are used to when working with real sheets of paper on a real desk, plus some more.

Moving Windows. Windows can be moved by pointing at the double arrow in the *title pane* and clicking the left mouse button. (The Symbolics mouse has three buttons.) After the click, an outline of the window will follow the movements of the mouse. Another left click deposits the window at the newly chosen location.

Reshaping Windows. Windows can be reshaped by clicking left on one of the arrows in the top right corner or top left corner of a window. After the click, the chosen corner will follow the movements of the mouse. Clicking left once more deposits the corner and makes the window change to the new outline. Figure 2 shows the screen after we have moved and enlarged Plot1. Note that the plot has automatically rescaled itself to fill up the larger window.

Burying Windows. A window can be buried by clicking left on the circular icon in the title pane. Burying a window means moving it to the bottom of the pile of paper on the desk, without changing its location.

Uncovering Windows. If a window is partially visible, we can expose it by pointing at its visible part with the mouse and clicking left. A completely hidden window can be retrieved by sequentially burying all the windows covering it up. Using folders, however (as described in Sec. 4), is usually a more efficient method for recovering a hidden window than simply digging through the pile.

Other Operations. There are several less frequently used actions that we want to perform on a window, like throwing it away or putting it into a folder. Other operations are specific to what the window shows. If it shows a scatterplot, for example, we want to add or remove coordinate axes. A list of such actions is presented in a menu that we obtain by clicking right on the title of the

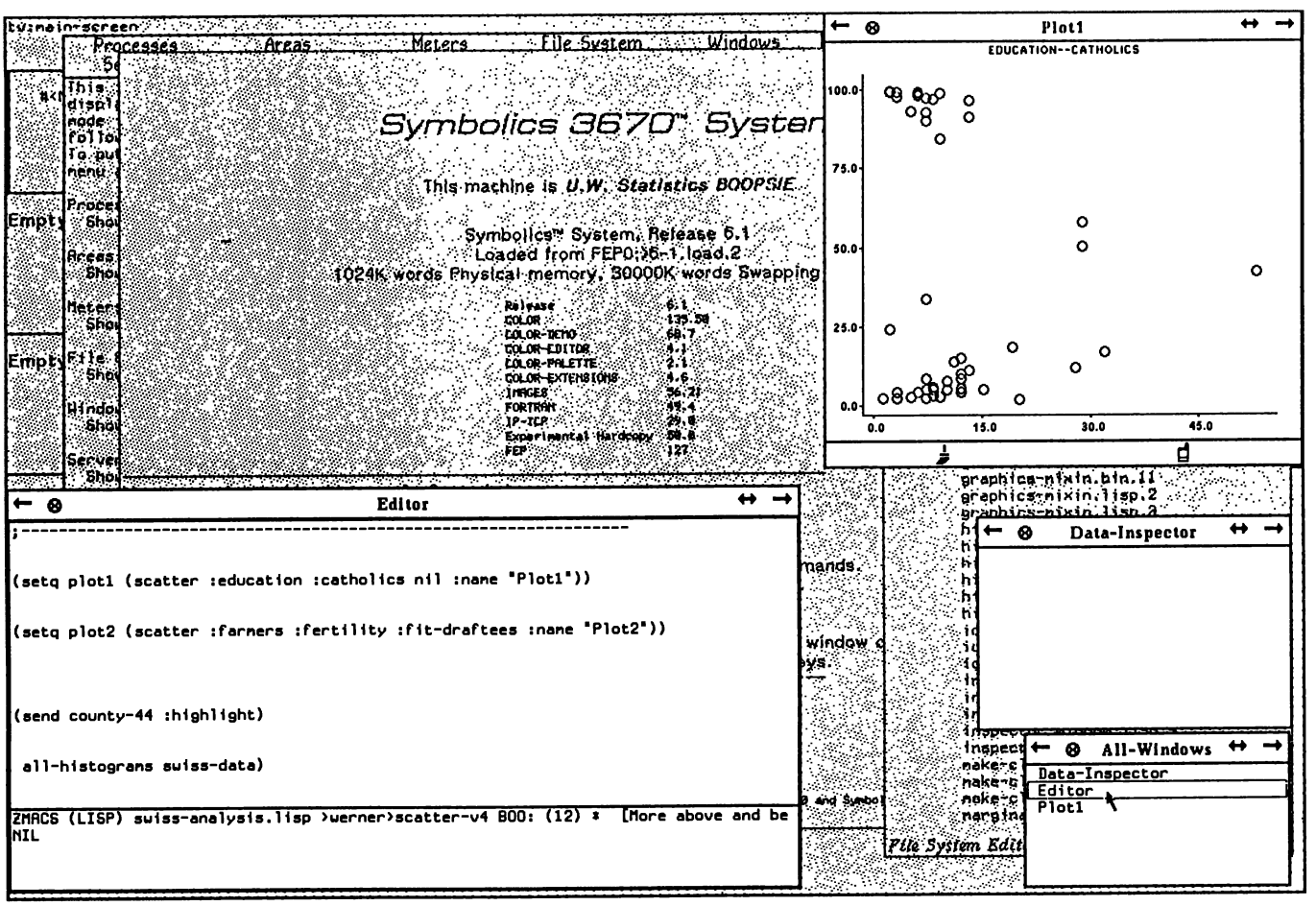

Figure 1. Desk Top.

window. We can then point at an action with the mouse and have it executed by clicking left.

4. FOLDERS

On the simplest level, a folder is just what we are used to—a place to collect (virtual) sheets of paper (i.e., windows). The title of every window in the folder is displayed in the folder's window. The text in a folder window can be scrolled up and down under mouse control if the window is too small to have room for all of the titles of the windows that it contains.

Folders have two important properties, which have no exact parallel in the world of real paper and cardboard. First, a folder can contain other Lisp objects besides windows, such as data sets or regression models. This is useful for organizing a data analysis and keeping track of the results.

Second, an object can be in more than one folder. Strictly speaking, a folder in our sense does not contain objects—it contains pointers to objects. This was done partly to avoid the notion of "copying."

The system maintains a list of folders, to which newly opened windows are automatically added. The user can create folders and add them to the list or remove folders from the list. This makes it easy to organize work and keep logically connected plots together. One folder, *All-Windows*, at the bottom right in Figures 1 and 2, is defined by default. It contains all of the windows that are currently in existence. Windows that are thrown away are automatically removed from all folders.

Clicking left on the title of a window in a folder alternately exposes and buries the window. On the Symbolics the response to the clicks is instantaneous. This makes it convenient to search for windows that got lost in the shuffle, providing an alternative to constantly inventing and remembering new, meaningful names. Clicking right on the name of a window in the folder brings up the same menu that we would obtain by clicking right on the name in the window's title pane.

5. A DATA SET

For the remainder of the article I will discuss scatterplots and histograms. To illustrate their capabilities, we will look at a set of demographic data for 47 Swiss counties in 1888. The data set was collected by Francine van de Walle of Princeton University to study the demographic transition occurring when Switzerland changed from a medieval agricultural society to a modern industrial society. It was reported by Mosteller and Tukey (1977). The variables are the following:

- *education*—proportion of population with education beyond primary school

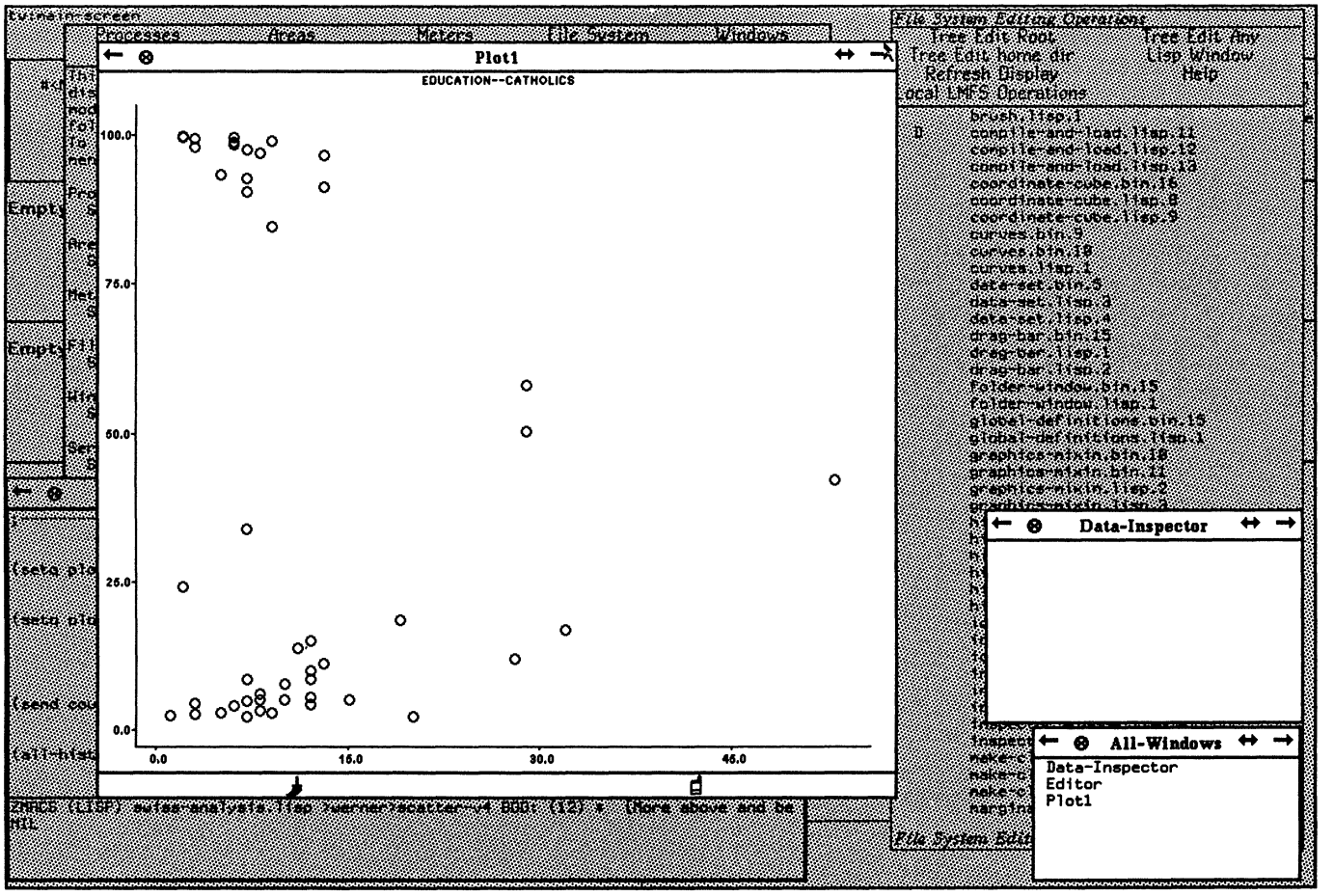

Figure 2. Desk Top After Moving and Enlarging the Scatterplot.

- *catholics*—proportion of Catholics
- *infant mortality*
- *fertility*—a measure of fertility
- *farmers*—proportion of inhabitants engaged in agriculture
- *fit draftees*—proportion of draftees receiving the highest mark on the army examination.

The *Plot1* window in Figures 1 and 2 shows a scatterplot of *catholics* on the vertical axis versus *education* on the horizontal axis. Internally, the data set is represented as a list of records, with one record for each observation. The fields of each record in our example are *:education, :catholics*, and so forth. In addition, there are fields specifying the label of the observation and the way in which the observation is to be drawn on the screen.

6. SCATTERPLOT WINDOWS

A scatterplot in Plot Windows differs from scatterplots produced by most standard statistical packages in three important aspects:

1. It can display an arbitrary number of 2-dimensional or 3-dimensional objects, such as point clouds, polygons, and coordinate axes.
2. It supports interactive operations.
3. It always reflects the current state of the objects that it displays. If these objects change, the plot automatically updates itself.

I will now discuss these three properties in more detail.

6.1 Displaying Objects

To give a feel for the type and level of commands, here is the Lisp statement that we have to type at the command window in order to create a new scatterplot window:

```
(setq plot2 (scatter :farmers :fertility :fit-draftees
                     :name "Plot2"))
```

This statement assigns the value returned by the *scatter* function to the symbol *plot2*; *setq* is the Lisp assignment operator, analogous to := in Pascal. The value returned by *scatter* is a Lisp data structure representing the scatterplot. (The *:name* parameter can be omitted, in which case the system makes up a default name.)

Figure 3 shows the screen after execution of the command. The new plot appears in a default size in the upper right corner. The proportion of inhabitants engaged in agriculture is drawn on the horizontal axis; *fertility* is drawn on the vertical axis. The third variable is not visible, because the third coordinate initially is orthogonal to the

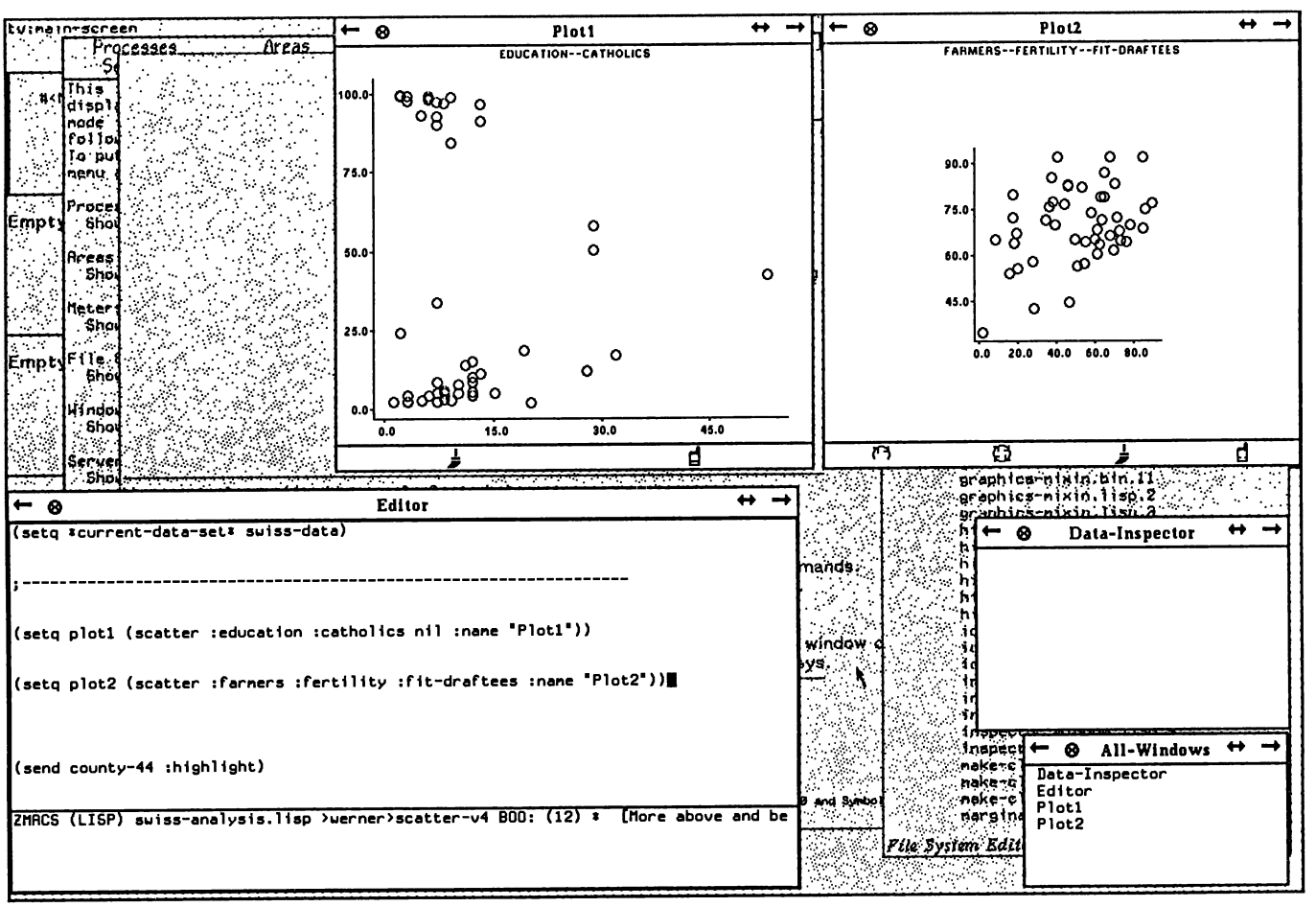

Figure 3. Desk Top With an Added Second Scatterplot.

screen plane. To perceive the 3-dimensional structure we have to rotate the point cloud, as described in Section 6.2.1.

In our example the *scatter* function is called with four arguments, the names of the three variables, and the name of the plot. We also could specify the name of the data set, in this case *swiss-data*. If no data set is specified, the default is used, which had been set to *swiss-data* previously. By giving additional arguments to *scatter*, the user can specify the locations of the top left and bottom right corners of the window or indicate that they are to be positioned with the mouse.

We can at any time add additional objects to a scatterplot, such as another point cloud, a curve, or coordinate axes, or we can remove objects from the plot. By default, scaling is automatic: the viewing transformation is chosen so that all objects in the display list are completely visible inside the window.

6.2 Interactive Operations

Scatterplot windows support a number of interactive operations. They are similar to the ones that were available in earlier systems for dynamic viewing of multivariate data, such as PRIM-9 and ORION.

6.2.1. Rotation. When a 3-dimensional object is shown in a scatterplot window, we initially see the projection of the object onto the first two coordinates. We can perceive the 3-dimensional structure by making the object spin and watching the projection on the screen change. Rotation is controlled either through the menu pane at the bottom of a scatterplot window or by typing commands at the *Editor* window.

Pointing at the leftmost icon in the menu pane and holding the left mouse button down makes the objects in the scatterplot spin around the screen's x axis. Rotation starts slowly and then accelerates to some maximum speed. Releasing the button stops the rotation; pressing it down again resumes the rotation, but in the opposite direction. In an analogous fashion, the middle and right buttons control rotation around the screen y and z axes, respectively. (The screen's z axis sticks out of the screen; rotation of a point cloud around the z axis moves the points in circles on the screen.)

Pointing at the next icon to the right and clicking left makes the objects spin around the x axis; in contrast to what was described before, however, the spinning continues after the button is released. Clicking left again stops the motion, and clicking left once more starts rotation in

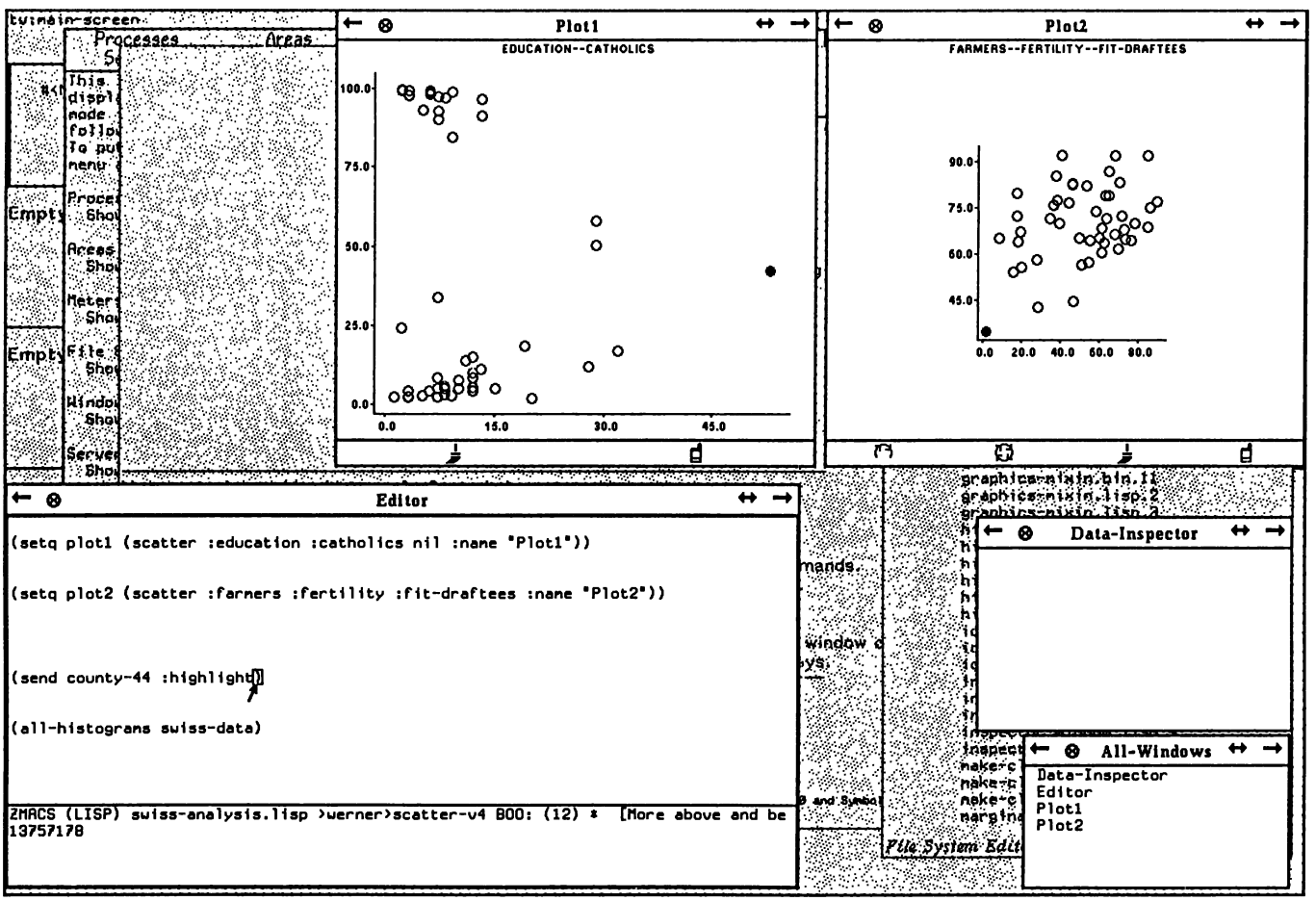

Figure 4. Desk Top After Highlighting Observation 44.

the opposite direction. Any command sent to a scatterplot window, either through menu selection or by typing at the command window, will also stop the rotation.

We can also make a scatterplot spin under program control. Evaluating the expression

(send plot2 :rotate-around :*y*-axis)

makes the objects spin around the *y* axis. Direction of rotation can be specified by providing an additional optional argument. The rotation can be stopped by sending the plot the *:stop-rotation* message. As a general rule, whatever can be done through menu selection can also be done under program control.

Of course we can have more than one scatterplot window on the screen at any time, and any number of windows can spin simultaneously. Spinning will also continue while new commands are executed or the user interacts with another plot. It should become apparent that this is a useful feature.

6.2.2. Identification and Highlighting. One of the most useful features of *Plot Windows* is the ability to establish a correspondence between observations and their representations on the screen. Our example does not illustrate this fact very well, because the observations have no meaningful labels—we do not know the names or locations of the 47 counties. Thus we labeled the observations simply by their sequence numbers.

To locate County 44 in the two plots, we issue the command

(send county-44 :highlight).

Figure 4 shows the screen after execution of the command. In each plot the icon representing County 44 has been highlighted. The command does nothing more than changing the *:intensity* field of the record representing County 44. This is enough to highlight the observation in all plots, because, by construction, plots always reflect the current state of the objects they display.

To get the label of any observation on the screen, we can click left on the pointing hand icon in the scatterplot menu and move the cursor inside the window. Pointing at any observation and clicking the left button highlights this observation, prints its label next to it, and displays the values of all of its variables in the *Data-Inspector*. Figure 5 shows the screen after labeling one of the observations in *Plot1*.

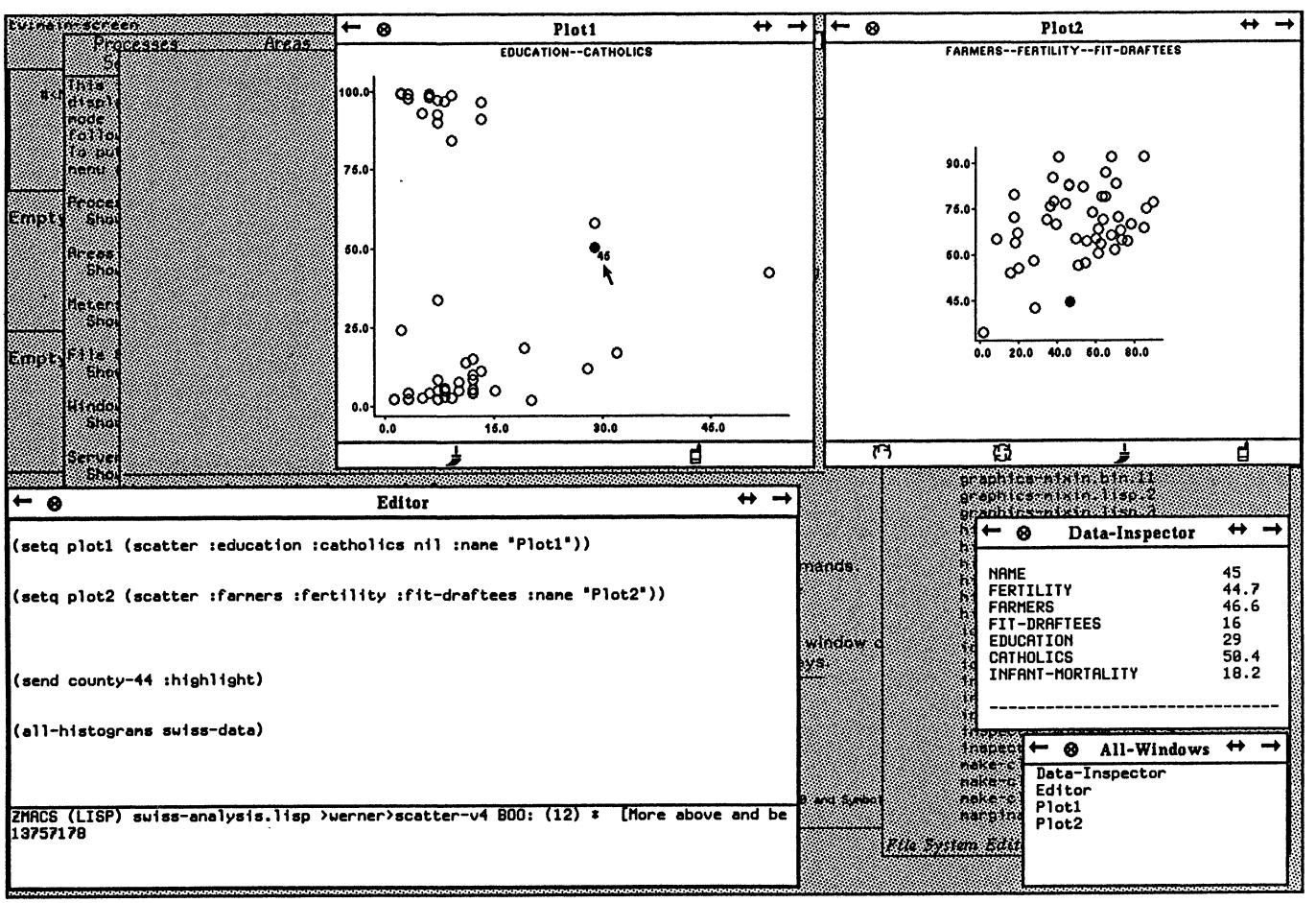

Figure 5. Desk Top After Identifying Observation 45.

6.2.3. Painting. We select the painting operation by clicking left on the paintbrush icon in the menu pane of a scatterplot window and moving the cursor inside the window. The arrow will be replaced by a little square, the brush, which we can move with the mouse. Painting with the left button held down highlights the observations that we brush over, whereas painting with the middle button held down removes highlights. The painting operation is terminated by moving the cursor out of the window.

The system offers a choice between different brush shapes, selectable from a menu. Apart from the square brush, there is a thin and high brush, stretching over the entire vertical range of the window, and a flat and wide brush, stretching over the entire horizontal range. We can also define our own brush by selecting the corresponding menu item and then positioning the upper left and lower right corners of the brush with the mouse.

Painting allows us to interactively define subsets of a data set. We simply paint the observations that we want in the subset and then evaluate a Lisp function that loops over the data set and collects into a list all those observations for which the *intensity* field has the value *:highlighted*.

6.3 Automatic Updating

A scatterplot always automatically reflects the latest state of the objects it displays. If the objects change, the scatterplot changes. This is a far-reaching concept. I will discuss here one particular application of this concept, connected scatterplots.

6.3.1. Connected Scatterplots. To illustrate the idea of connected scatterplots, let us return to Figure 3. It shows a plot of *catholics* versus *education* and a plot of *fertility* versus *farmers*. These two plots, however, do not give a complete insight into the distribution of the data in the 4-dimensional space spanned by the 4 variables involved; we cannot reconstruct the 4-dimensional coordinates of the points from the 2-dimensional projections. For the reconstruction we also need to know which point in one scatterplot corresponds to which point in the other—we need to be able to identify pairs of points corresponding to the same observation. This connection can be established through painting and automatic updating. We select the painting operation in one of the scatterplots (the *active plot*) and highlight or downlight observations. All other scatterplots automatically keep themselves up to date. Thus the intensity with which an observation is drawn is always the same in all plots showing the observation.

Automatic updating of scatterplots to show connections between points was first suggested and implemented by McDonald (1982). He used transient highlighting based on distance—at any time points within a certain distance from the cursor and the corresponding points in the other plots were highlighted. The idea of painting, that is, the

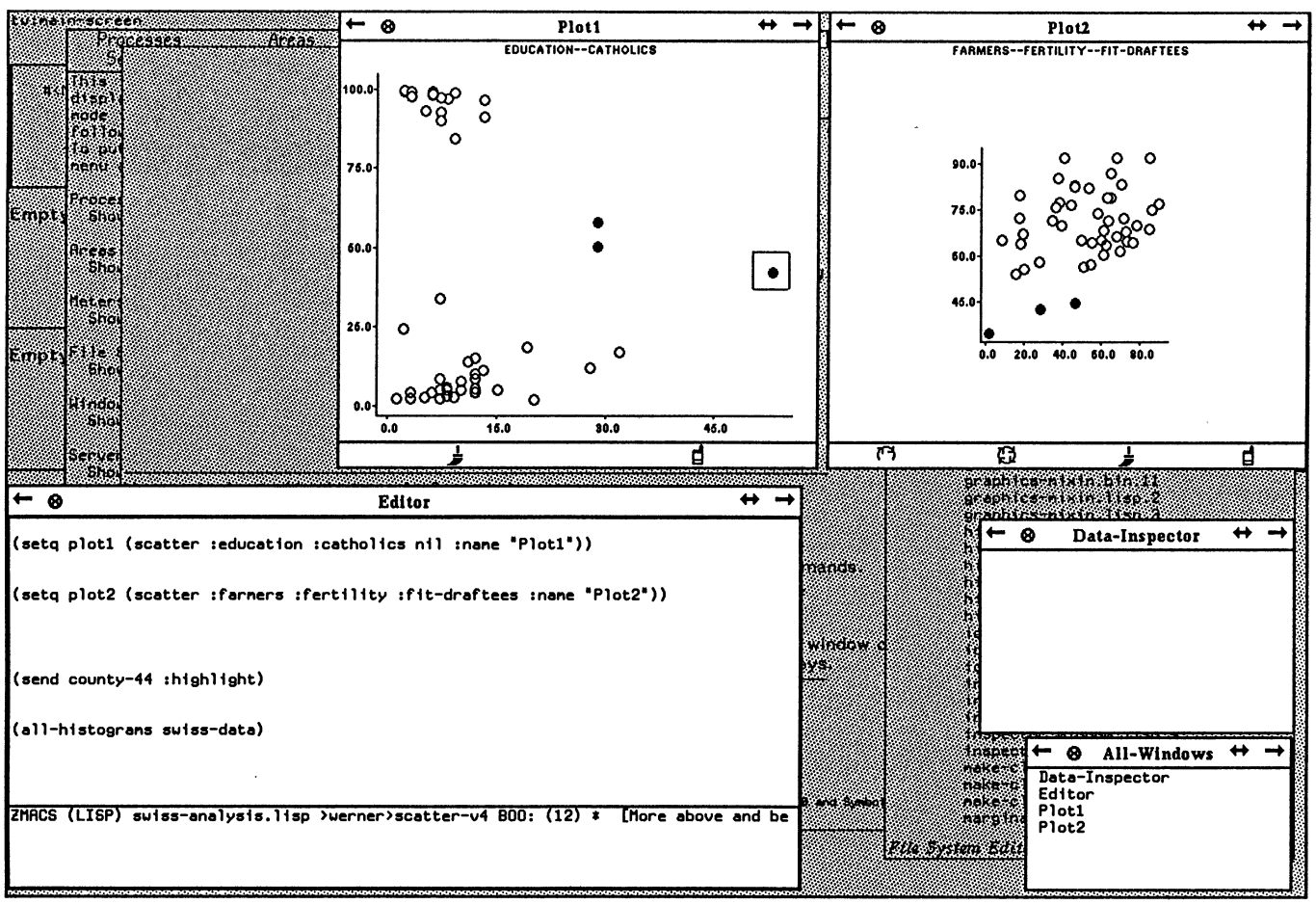

Figure 6. Desk Top After Painting of the Outliers in Plot1.

persistent highlighting described in Section 6.2.3, and of using brushes of various shapes, is due to Becker and Cleveland (1985).

6.3.2. Illustration. To illustrate the usefulness of connected scatterplots, let us have a closer look at our data set. We first highlight the three isolated points in *Plot1* by painting them with the left mouse button held down. Figure 6 shows that those three districts are also extreme in *Plot2*—they have the lowest values of *fertility*.

We would also like to know where the two clusters with a high proportion of *catholics* and a low proportion of *catholics* fall in *Plot2*. We first erase the highlights by painting over the highlighted observations with the middle button held down. Figure 7 shows the screen after highlighting the cluster with a high proportion of *catholics*.

We see in *Plot2* that districts with a high proportion of *catholics* also have a high proportion of *farmers* and high *fertility*. Spinning the cloud in *Plot2* shows that they also tend to have a low proportion of *fit draftees*. Note that we can spin one scatterplot while painting another; this ability is useful, allowing us (at least in theory) to completely reconstruct the geometric configuration of 5-dimensional data. Districts with a low proportion of *catholics* (the lower cluster in *Plot1*) tend to have lower *fertility* than the districts with a high proportion of *catholics*. The three outliers do not follow the rule—they are about 50% Catholic, but they have the lowest values of *fertility*.

6.3.3. Engineering Aspects of Painting. Painting taxes the computer hardware. A substantial number of plots besides the active plot can be exposed on the screen, and they all have to be updated constantly. (We will see an example with eight plots.) Processor time has to be divided between moving the brush and changing the intensities in the active plot, and bringing the other plots up to date. Experience shows that it is crucial for the brush to follow smoothly and continuously the mouse—jerky brush movement is highly disturbing. This was resolved by halting any other activity in the system during painting: While the mouse is moved with either the left or the middle button held down, other plots are not updated, all rotations stop, and so forth. As soon as painting is interrupted (all mouse buttons released), the other plots are updated, rotations start again, and system activity reverts to normal. The fact that other plots are momentarily out of date and rotations are stopped does not seem to matter, because during painting one is forced to concentrate on the active plot; it is very hard to look at two plots simultaneously.

There are two slightly different ways of bringing the other plots up to date, once painting is interrupted. Either we can erase them totally and then redraw them from

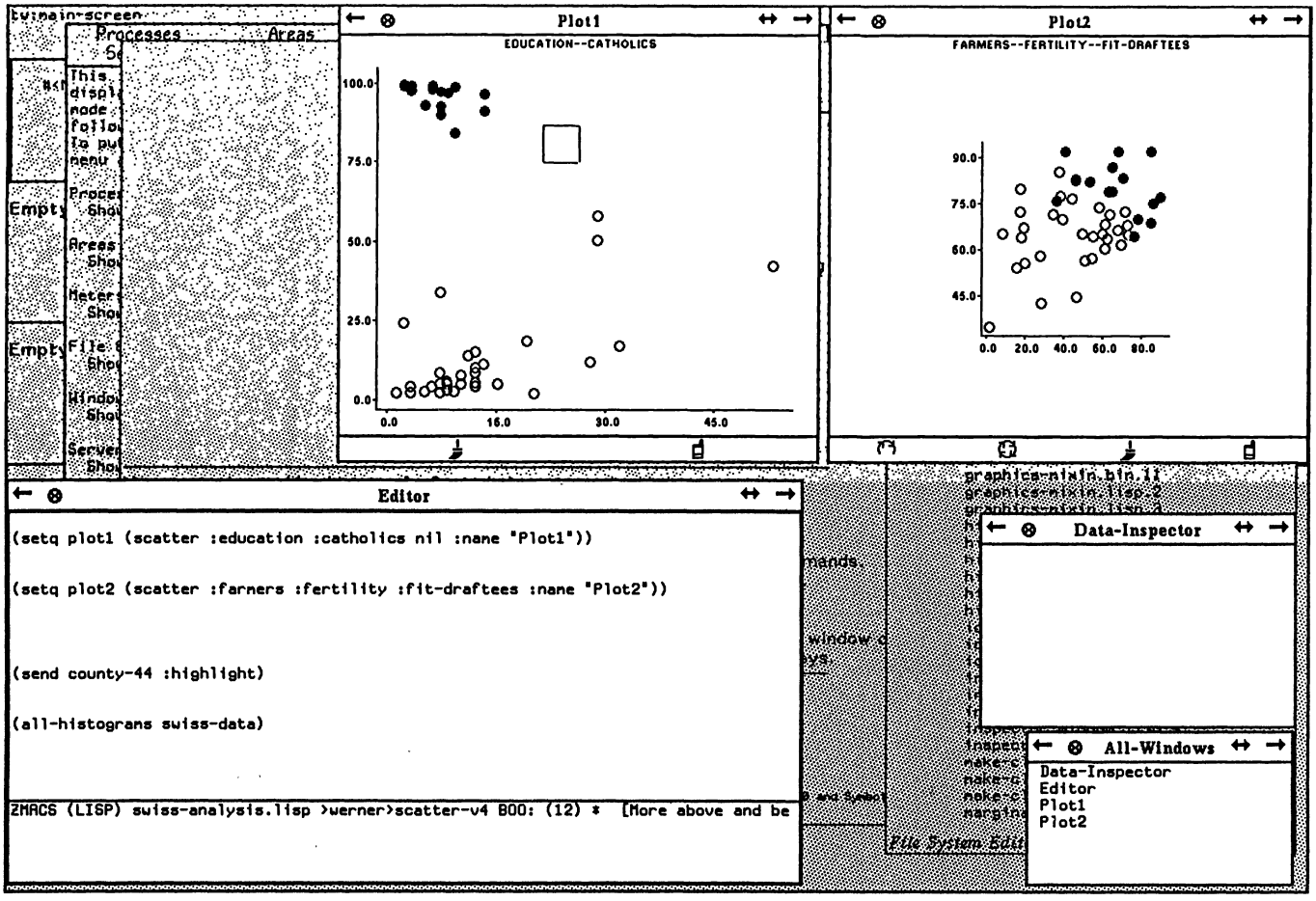

Figure 7. Desk Top After Painting of the Cluster With a High Proportion of Catholics.

scratch, or we can incrementally update the icons representing the observations whose intensities have been changed. Both ways were tried, and it was found that the second one is clearly preferable. Suppose we have two scatterplots on the screen, and *Plot1* is the active plot. We will want to see which icons in *Plot2* change intensity, once we interrupt painting. This is difficult if *Plot2* is completely erased and then redrawn. Our memory of the picture apparently does not survive the minute period during which the window is blank.

The preceding remarks illustrate the fact that small details in the implementation of interactive methods can make a big difference in ease of use. Therefore, it is important to build prototype systems in an environment that allows one to make changes quickly and easily.

7. HISTOGRAMS

Discovering structure in a data set helps in our understanding only if we can describe what we see in terms of the underlying variables. To give a concrete example, suppose we spin a 3-dimensional scatterplot and we see a cluster. We then need to come up with a statement like "These districts have medium proportion *catholics,* low *fertility,* and high *education.*" Extracting such a statement, based on a 3-dimensional coordinate system drawn into the plot, requires considerable mental effort. Depending on the orientation, the axes can crisscross the point scatter, cluttering it up, and it is always difficult to see whether they point into the screen or out of the screen.

As a first step in describing structure, it is useful to be able to characterize an observation or group of observations in terms of the original variables, such as "The three outliers correspond to districts with the lowest *fertility,* medium proportion *catholics,* highest *education,* the highest proportion of *fit draftees;* they are not special in terms of *infant mortality.*" As long as each of the variables is represented in at least one scatterplot, we have the necessary information on the screen. We just have to paint the observations in question and then see where they fall in the other plots. Note, however, that the preceding statement concerns only 1-dimensional marginals, and it seems a waste of mental effort to extract it from plots of 2-dimensional marginals.

These two empirical observations suggest implementing histograms. A histogram in *Plot Windows* is special in three ways:

1. Each bar of the histogram is composed of the icons representing the observations that fall into the corresponding class. In this aspect, the histogram resembles a stem-and-leaf plot.

2. The intensities of the observations are automatically kept up to date. The highlighted observations in each class

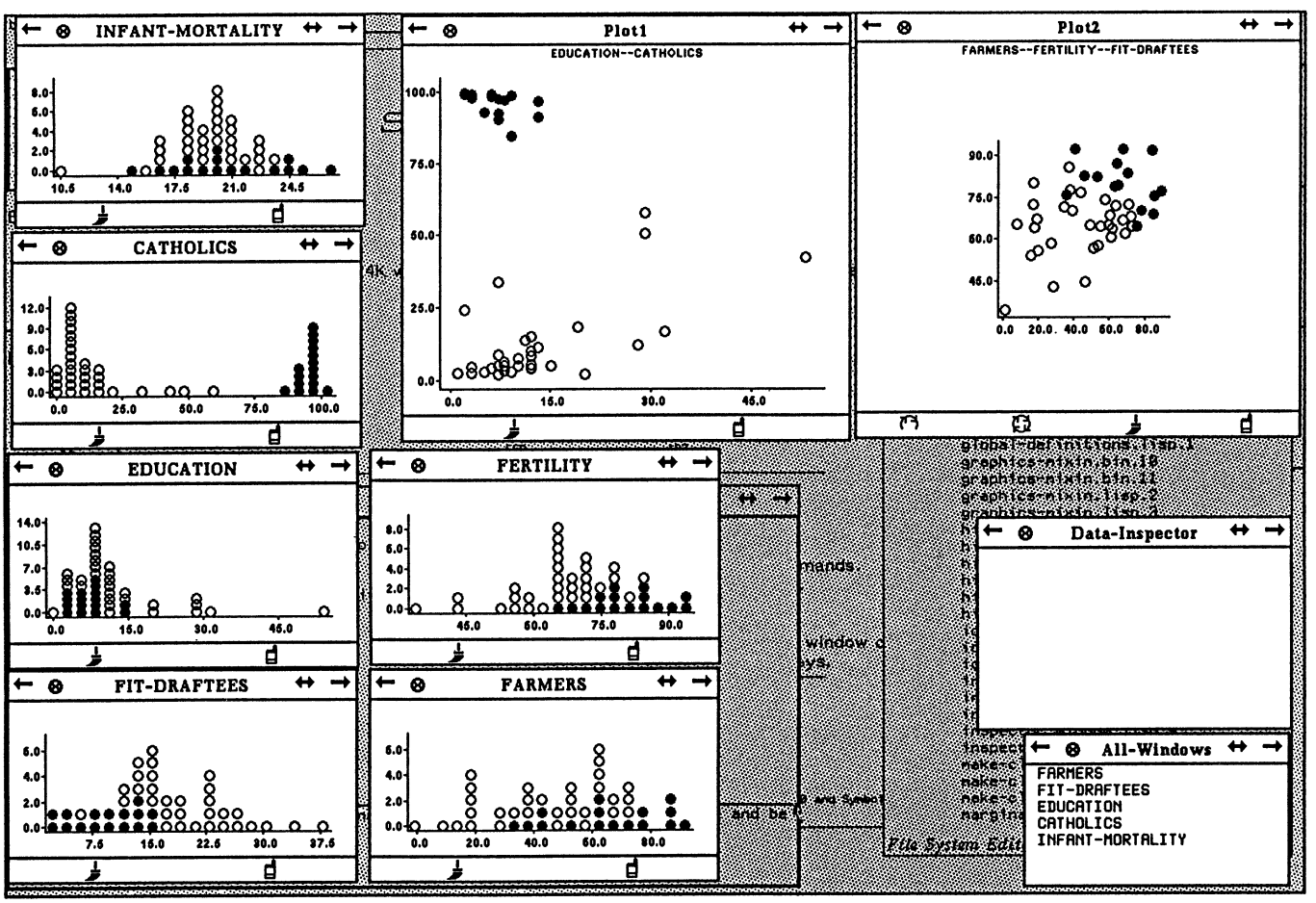

Figure 8. Desk Top With Marginal Histograms.

are always drawn at the bottom of the corresponding bar, and the others are drawn at the top. This arrangement makes it easy to see what fraction of the observations in a particular bar is highlighted.

3. Like a scatterplot, a histogram supports the *identify* and *paint* operations. We can paint outliers and clusters apparent in a 1-dimensional marginal and see where those observations fall in the other variables. We can also easily answer questions like "What goes with a high proportion of *catholics?*" by simply painting the right tail of the *catholics* histogram.

Figure 8 shows the screen with the two scatterplots and histograms of the six variables. Note that the location of the Catholic districts in each of the variables is now readily apparent.

8. DISCUSSION

The *Plot Windows* system—which allows the data analyst to view, manipulate, and store a large number of plots—offers a major increase in the ability to dissect and interpret multivariate data. Painting of connected scatterplots and histograms appears to be more useful for detecting and understanding structure than rotation, because it is not limited to three dimensions. There have been many attempts at displaying more than three dimensions in a single plot. A rather natural idea for showing additional variables is to encode them in the shape, color, or intensity of the icons representing the observations. But extracting a statement like "high values of variables 1 and 3 go with low values of variables 2 and 4" from a spinning cloud of icons is not nearly as immediate as grasping and describing a spatial point pattern.

A different approach to presenting higher-dimensional data is the *grand tour,* a dynamic display of the projections of a high-dimensional point set on a continuous sequence of 2-dimensional planes (Asimov 1985; Buja and Asimov 1985). Watching a spinning 3-dimensional point cloud means watching a special grand tour of 3-dimensional space: one of the vectors spanning the projection plane remains fixed; the other one rotates. In more than three dimensions, however, we are not able to synthesize the sequence of views into a mental image of the configuration; our ability to visualize high-dimensional space is too limited. The *grand tour* is a useful tool for detecting clusters and outliers in higher dimensions; it appears to be less suitable for deriving statements about the associations between variables.

Luckily, in an environment like the one described here we do not have to decide whether we want to look at marginal histograms, at scatterplots, or at a grand tour—we can have it all simultaneously, and we can connect what we see in the various plots through painting. Higher-dimensional views show us features of the data that might not be visible in lower dimensions; lower-dimensional views, on the other hand, are easier to interpret. We can have the advantages of both at the same time.

[*Received June 1986. Revised November 1986.*]

REFERENCES

Asimov, D. (1985), "The Grand Tour: A Tool for Viewing Multidimensional Data," *SIAM Journal on Scientific and Statistical Computing,* 6, 128–143.

Becker, R. A., and Chambers, J. M. (1984), *S: An Interactive Environment for Data Analysis and Graphics,* Belmont, CA: Wadsworth.

Becker, R. A., and Cleveland, W. S. (1985), "Brushing a Scatterplot Matrix: Graphical Methods for Analyzing Multidimensional Data," preprint, AT&T Bell Laboratories, Murray Hill, NJ.

Buja, A., and Asimov, D. (1986), "Grand Tour Methods: An Outline," in *Computer Science and Statistics: Proceedings of the 17th Symposium on the Interface,* Amsterdam: Elsevier.

Donoho, A. W., Donoho, D. L., and Gasko, M. (1985), *MacSpin Graphical Data Analysis Software,* Belmont, CA: Wadsworth.

Donoho, D., Huber, P. J., Ramos, E., and Thoma, H. (1982), "Kinematic Display of Multivariate Data," in *Proceedings of the Third Annual Conference and Exposition of the National Computer Graphics Association,* Fairfax, VA: National Computer Graphics Association.

Fisherkeller, M. A., Friedman, J. H., and Tukey, J. W. (1974), "Prim-9: An Interactive Multidimensional Data Display and Analysis System," Publication 1408, Stanford Linear Accelerator Center, Stanford, CA.

Fowlkes, E. B. (1971), "User's Manual for an On-Line Interactive System for Probability Plotting on the DDP-224 Computer," technical report, Bell Telephone Laboratories, Murray Hill, NJ.

Friedman, J. H., McDonald, J. A., and Stuetzle, W. (1982), "An Introduction to Real Time Graphics for Analyzing Multivariate Data," in *Proceedings of the Third Annual Conference and Exposition of the National Computer Graphics Association,* Fairfax, VA: National Computer Graphics Association.

McDonald, J. A. (1982), "Interactive Graphics for Data Analysis," unpublished Ph.D. thesis, Stanford University, Dept. of Statistics.

McDonald, J. A., and Pedersen, J. (1985a), "Computing Environments for Data Analysis, Part 1: Introduction," *SIAM Journal on Scientific and Statistical Computing,* 6, 1004–1012.

——— (1985b), "Computing Environments for Data Analysis, Part 2: Hardware," *SIAM Journal on Scientific and Statistical Computing,* 6, 1013–1021.

——— (in press), "Computing Environments for Data Analysis, Part 3: Programming Environments," *SIAM Journal on Scientific and Statistical Computing.*

Mosteller, F., and Tukey, J. W. (1977), *Data Analysis and Regression,* Reading, MA: Addison-Wesley.

PC-ISP (1985), *Interactive Scientific Processor User's Guide and Command Descriptions,* London: Chapman & Hall.

Ryan, T. A., Joiner, B. L., and Ryan, B. F. (1976), *Minitab Student Handbook,* North Scituate, MA: Duxbury Press.

Stuetzle, W. (1986), "Design and Implementation of Plot Windows," in *Proceedings of the Statistical Computing Section, American Statistical Association,* pp. 32–40.

Velleman, P. F. (1986), "Graphics for Specification: A Graphic Syntax for Statistics," in *Computer Science and Statistics: Proceedings of the 17th Symposium on the Interface,* Amsterdam: Elsevier, pp. 59–62.