NASA CONTRACTOR REPORT

NASA CR-1628

INTERACTIVE SPECIFICATION OF DATA DISPLAYS

by R. Anderson, R. Bator, R. Gagan, J. Meads, and L. Metrick

Prepared by
WOLF RESEARCH & DEVELOPMENT CORPORATION
West Concord, Mass. 01781
for Goddard Space Flight Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JUNE 1970

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-1628		

4. Title and Subtitle
INTERACTIVE SPECIFICATION OF DATA DISPLAYS

5. Report Date
June 1970

6. Performing Organization Code

7. Author(s) R. Anderson, R. Bator, R. Gagan, J. Meads, L. Metrick

8. Performing Organization Report No.
None

9. Performing Organization Name and Address
Wolf Research & Development Corporation
P. O. Box 36 (Baker Avenue)
West Concord, Massachusetts 01781

10. Work Unit No.

11. Contract or Grant No.
NAS5-9756-47C

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, D.C. 20546

13. Type of Report and Period Covered
Contractor Report

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The purpose of this research effort was to postulate and experiment with new techniques for interactively creating data displays. The approach discussed in this report was to explore a auser oriented language for specifying data display procedures as opposed to data manipulation procedures that could be specified using conventional computer languages such as FORTRAN or ALGOL.

In accomplishing this effort the appropriateness of using a list-or string processing language for programming the users language is discussed. A set of data display language operators is proposed and a subset of these were programmed for a PDP-1 with COLOR-CRT display using an interpretive language. Experiments were conducted to obtain user reaction to the concept. Finally, a formal specification of the data display language syntax rules is developed.

The conclusions reached are that such a users oriented display language is relatively easy to use and meaningful work can be accomplished by combining a few basic operators to form display procedures. Further research in this area should consider the inclusion of mathematical expressions within the data display language.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Data display, computer graphics, display programming	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
(U)	(U)	68	$3.00

*For sale by the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.

1, Display Systems

ACKNOWLEDGMENTS

This work was performed and the documentation prepared under the technical direction of T. P. Gorman and D. E. Jamison, GSFC Code 565.

Color computer display equipment and programs were made available for use in the development of the pilot test system by Air Force Cambridge Research Laboratories, through the courtesy of Charlton M. Walter, Chief, Dynamic Processes Branch.

Principal individual contributors to this effort included:

R. Anderson
R. Bator
R. Gagan
J. Meads
L. Metrick

H. T. French
Project Manager

CONTENTS

CONTENTS (Continued)

ILLUSTRATIONS

ILLUSTRATIONS (Continued)

1.0 INTRODUCTION

This work is to be consistent with the results of research studies and experimental investigation performed under GSFC task order NAS5-9756-47, 47A, and is directed toward the advancement of interactive software tools for use by the non-programmer in operating upon and displaying large volumes of data.

In "A Report on Data Display Programming,"[1] a software system was postulated that would enable a non-programmer to develop data analysis strategies by using an on-line graphical language. The current effort can best be explained by comparing it with previous research in the field of computer graphics.

The Sketchpad Language[2] was graphical in the sense it enabled one to define pictorial information. It was not a graphical language in the sense that it used pictorial information to convey the meaning of the elements of the language.

The work of W. R. Sutherland[3] was a true graphical language in the sense that the elements of the language were graphical figures. Connections and positional relationships of the figures specified the procedure to be performed. The types of operators were analogous to machine language instructions and the capability to build and name procedures was included.

The development of an interactive display language for constructing data analysis strategies to analogous to the latter effort discussed above. One would like to be able to graphically specify the procedures for processing and displaying a set of data. Ideally, the language should be problem-oriented so that the user can devote the majority of his effort toward understanding the phenomenon represented by his data. This means he will not have to learn an extensive set of rules of a programming language or be required to have more than a cursory knowledge of computer structure and operation. Further, the language should fully reflect the power of graphical information in the syntax of the language, as well as the representation of results of processing specified by the language.

The current research effort is to define a graphical language for conducting data analysis experiments. This includes specifying the syntax in a way that makes it easily understood to the user. The elements of the language should be of such a level that the user will not have to work at specifying many routine details for each procedure.

2.0 PROBLEM DESCRIPTION

In this section the scope of work to be accomplished and the guiding constraints are discussed. The reader is advised that knowledge of Section 4 of "A Report on Data Display Programming"[1] would provide further insight to the following discussion.

2.1 A USER-ORIENTED DISPLAY LANGUAGE

The term "Display Language" for use in this discussion is defined as a set of symbols generated on the surface of a graphical display (CRT) console whose meaning is understood by the user and the display/computer system. The attractiveness of such a language has been previously stated by L. G. Roberts:[4] "A graphical language is tremendously powerful because it is a natural form of human representation and it derives richness and economy from its multi-dimensional character."

The problem of developing an appropriate display language for data analysis is somewhat analogous to specifying a special language for a syntax directed translator which would accept the display language primitives and perform the desired operations. The specification of the primitives must take into account such things as the type of problem, the potential user's background, and the level of detail at which the user will be required to work.

2.2 DESIGN CONSTRAINTS

The primary constraints for this work is that of the user having single input device and a single display device. By further restraining the use of these two devices to a common working surface, the user can only indicate his desires by pointing to or "picking" information if the display device is of the nature of a cathode-ray tube. The display primitives previously proposed[1] were designed with this constraint in mind.

2.3 SCOPE OF WORK

The primary purpose of the current effort is the investigation of software techniques to facilitate the generation and manipulation of data displays. In the previous effort a complete software system was postulated as shown in Figure 1. The primitive and control operators covered the entire range of functions to be performed including: 1) the specification of data displays, 2) the specification of data analysis sequences, 3) the ability to create special characters, 4) the documenting of results obtained from data analysis, 5) the specification of data to be analyzed, 6) the application of data

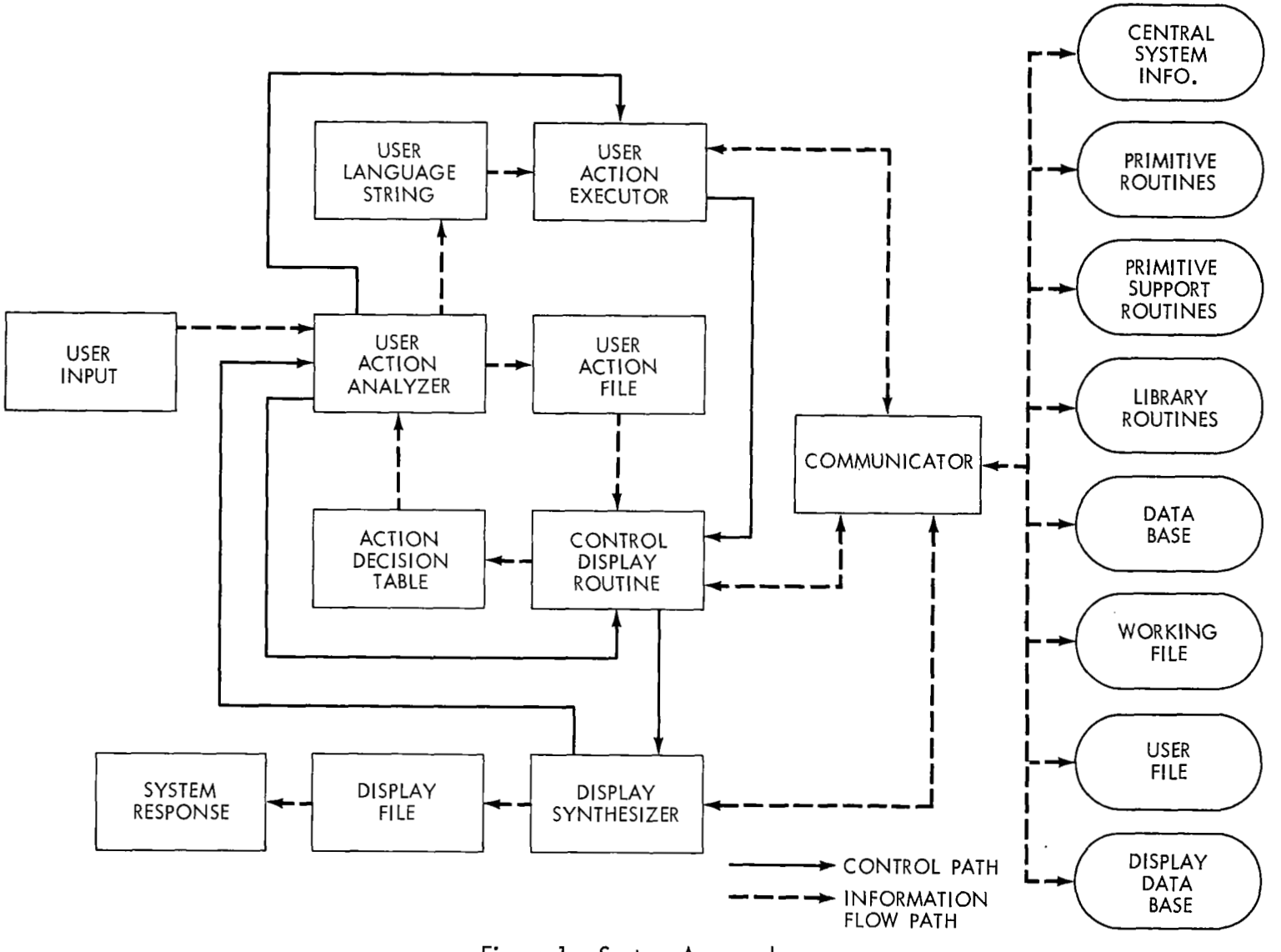


Figure 1. System Approach

analysis strategies to specific data sets, 7) the manipulation of data displays, and 8) various bookkeeping functions such as sign-on and sign-off procedures. The current effort will be concerned only with a portion of Items 1) and 7).

There are three basic questions to be considered in the investigation of software techniques for graphical input/output devices and the on-line specification of procedures via a graphical language.

The first question is one of determining the syntax of the user's language. The goals are a flexible, self-explanatory, easy-to-use language. In Section 3 of this report a set of display specification and manipulation operators are documented.

The second question is what type of language should be used to program an on-line interactive graphical display system. The research community in the field of graphics has expended a great deal of effort in exploring list and string processing languages for their application to graphics problems. A portion of the current effort includes a discussion of the use of such languages with respect to the on-line data analysis system and is presented in Section 4.

The third and equally important question is on the subject of data structuring. This subject was discussed in a general way in the previous effort, and the intention here is to extend that work in a specific and more detailed discussion of the subject. In Section 5 of the report the advantages of different techniques are presented along with a recommended data structure design for data display and manipulation.

The ideas and techniques discussed above were pilot tested on the Air Force Cambridge Research Laboratories Experimental Dynamic Processor (DX-1) with color display console. The pilot testing included the programming of a portion of the display language operators making use of the results of the language and data structure studies. An experimental evaluation was conducted using the display language operators, and the results of this evaluation is discussed in Appendix II.

3.0 DISPLAY LANGUAGE

This section contains a detailed discussion of the primitive operators that the experimenter will be able to use to construct and manipulate graphical data displays.

3.1 GRAPHICAL DISPLAY SPECIFICATION

A sub-mode of create display mode is set graph mode, which selects, through user options, the initial form of graphing the user's data. This mode is an intermediate step between raw data and dynamic, visual data manipulation.

The user options of this mode are indicated to the system by pointing at various light symbols on the bottom portion of the scope, i.e., the area for the temporary operators unique to this mode. At present the select graph mode has two main sections: two-dimensional and three-dimensional graphs. The only difference between the two is the addition of a few operators and a few more user specifications required. Therefore, the two will be discussed together.

Selecting this mode will present to the user a set of special operators for this mode (see Figure 2). Description of operators follows:

This operator will indicate that a two-dimensional graph is to be constructed.

This operator will indicate that a three-dimensional graph is to be constructed.

NAME

After selecting this operator the user will spell out the name by which he wants to refer to the graph he is creating.

DEFF: Description of Quantities to be Plotted

Serious data manipulations which extract more sophisticated data from the raw data is in the field of numerical analysis. Since this is beyond the scope of this project, certain assumptions have been made.

Assume a system user has a set of data of the form $(X_1, \ldots, X_n)$ where each X_i is a specific measurement. Otherwise, the data is unorganized. The user wishes to display his data as a two- or three-dimensional graph.

Each set of data will have its own name. Therefore it is a simple matter of issuing a command to plot and specifying which components of the vectors making up the data set should be used as coordinates. Then the system will search through the data set and display a point for each datum which has the specified components within the range of the graph.

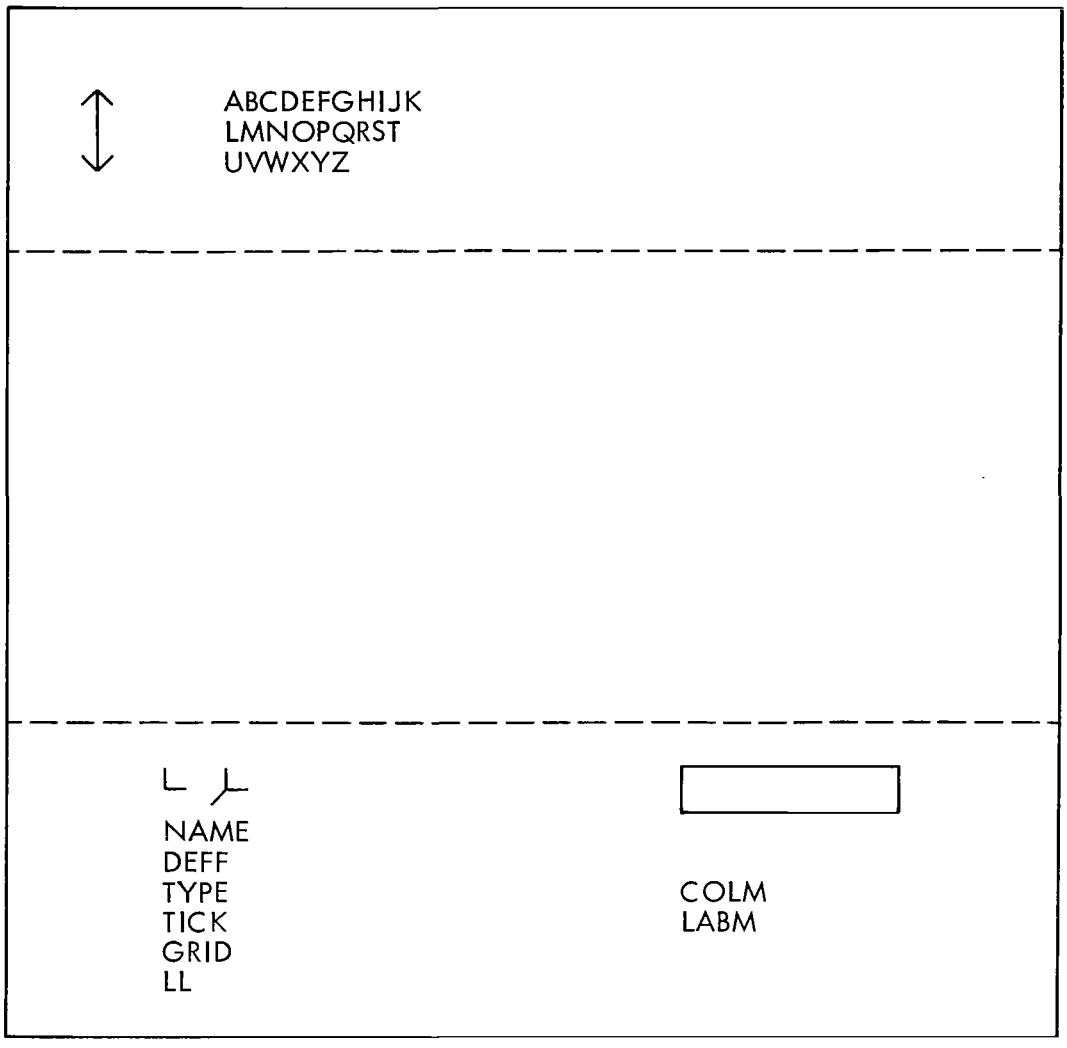


Figure 2. Set Graph Mode Control Operations

For example, assume we have a data set called PARCT, which consists of four components: distance from earth, particle count, magnetic field amplitude, and time. These components would be respectively referred to as PARCT(1), PARCT(2), PARCT(3), and PARCT(4). Now if a user wished to plot particle count versus distance from earth, he could issue the command, PLOT PARCT(1, 2). The graph generated would have particle count, PARCT(2), as the vertical axis and distance from earth, PARCT(1), as the horizontal axis.

To plot magnetic field amplitude versus distance from earth and time, the command PLOT PARCT(3, 1, 4) would generate a three-dimensional rectilinear point plot.

Other graphs such as plots of functions and surfaces could be generated, but the form of the function definition and the data ordering are too closely connected with numerical analysis to be able to describe an effective system at this time.

3.1.1 Range of axis in graphical display. After deciding which quantities are to be plotted, the user is given range specification commands to complete. These are:

SET X (min, max)
SET Y (min, max)

and for 3D plots

SET Z (min, max)

where the underlined portion is filled in by the user. Min and max are numbers specified by the user and represent the smallest and largest numbers, respectively, to be plotted on the corresponding axis.

Scaling to associate the values to the length of the axis is done automatically by the system, e.g.:

Set X-axis from 0.0 to 100.0
Y-axis from 200.0 to 400.0
Z-axis from -1.0 to 1.0

The following commands would be used:

SET X (0.0, 100.0)
SET Y (200.0, 400.0)
SET Z (-1.0, 1.0)

where the underlined portion is filled in by the user.

TYPE

This specifies type of graph desired. Pointing at "TYPE" will give the user four options to choose from: dot, line, constant

surface or level contour graph (see Figure 3). Note: two-dimensional graphs will have line fitting segments instead of constant surface and level contour.

TICK

The operator will give the user the option to specify the number or spacing of tick marks on each axis (see Figure 4). The user will select an "*" and then a number to correspond.

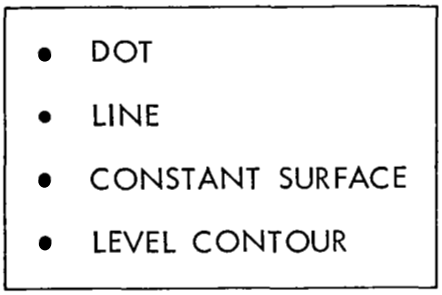


Figure 3. Graph Type Options

	AXIS	NO.	SP.
	X	*	*
	Y	*	*
(ONLY FOR 3-DIM)	Z	*	*

Figure 4. Tick Mark Selection

GRID

This operator constructs a planar grid between 2 axes (see Figure 5). The grid is spaced on tick marks.

LL

Selecting this operator means that a scale other than all linear axes is desired (see Figure 6). E.g., by pointing to "e" in the second row, the Y axis will have natural logs as its scale.

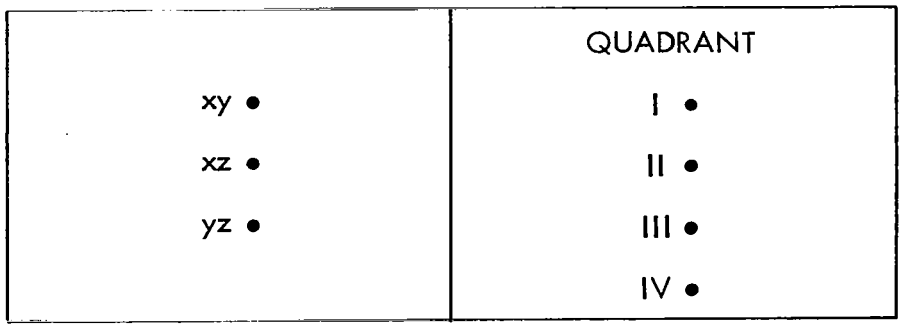


Figure 5a. 3-Dim. Choices Figure 5b. 2-Dim. Choices

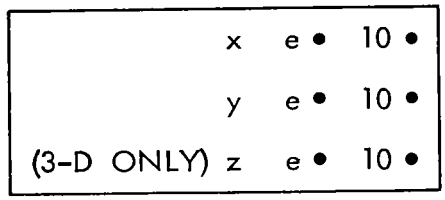


Figure 6. Types of Log Axes

COLM

This operator will provide additional operators for color coding of data and background information.

Normally a displayed graph of a data set will appear entirely in a single color. The COLM operator will provide additional control operators for the user to specify the color coding he desires of any portion of the graph (i.e., axes, data, title, background grid, etc.). It is also possible to have the color of a data set vary based upon the relative magnitude of the data values or to color code individual data sets with a unique color or range of colors.

Colors are displayed in the upper control portion of the CRT in two ways (see Figure 7). First, a series of dots will be visible which will each pertain to a different color (red, blue, green, magenta, cyan, yellow, orange, white, etc.). Second, a line will be seen which resembles the visible spectrum and contains all colors and shades available to the CRT. By picking a dot or a point on a line, an object on the graph, and then the ASSOCIATE button (see Figure 7), the object that was picked will be given a new color. An object can be ASSOCIATE'd as often as desired, thus eliminating errors as to choice of color or object.

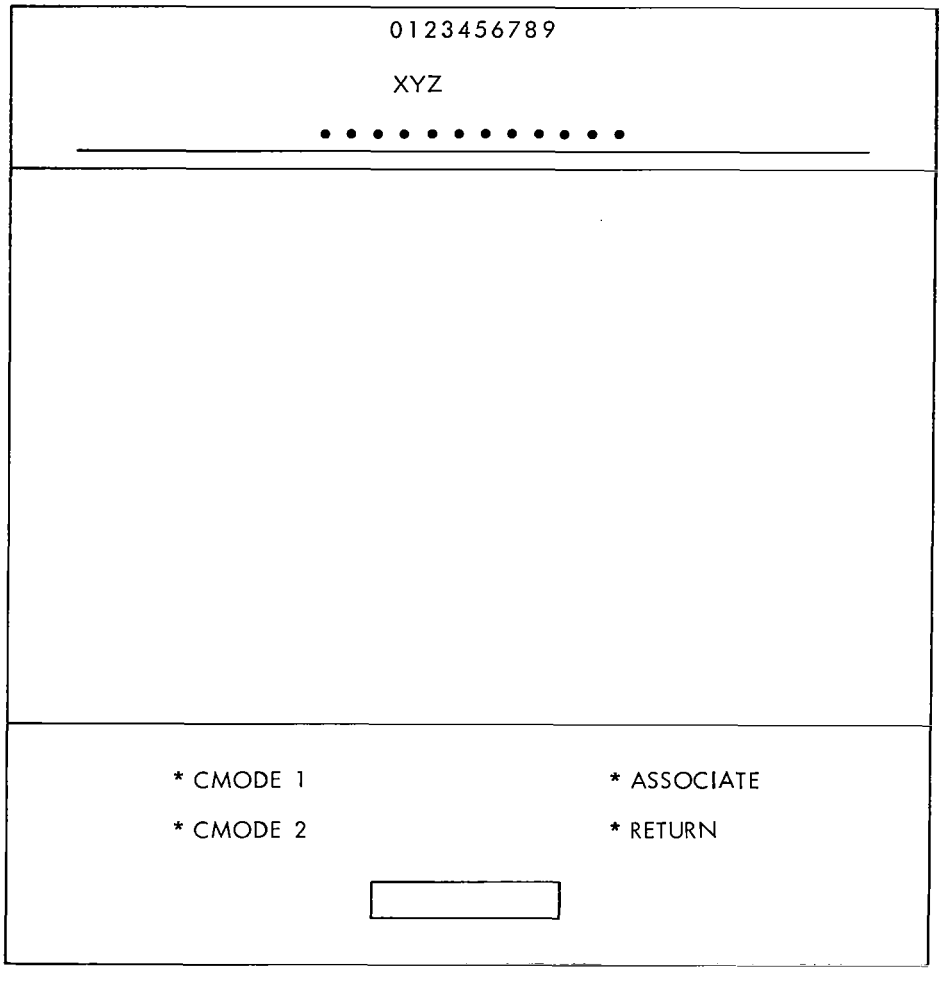


Figure 7. Color Sub Mode Options

In order to have the data color coded based upon the data values, the user would pick either two dots or two points, the first color being related to the minimum value along a chosen axis* and the second color related to the maximum value. (These values were specified in the SET-GRAPH mode.) He then points to the numbers in the upper control portion which then appear in the register, specifying the number of different colors along the axis.

The most natural manner for specifying color seems to be in terms of hue (wavelength), brilliance (whiteness), and saturation

*Axis is chosen by picking X, Y, or Z in upper control portion.

(purity of color), as shown in Figure 8A. Furthermore, one or more equipment manufacturers are currently developing equipment which will be programmable in these terms. This mode of color specification will therefore be considered the primary mode and will be called by activating the operator CMODE 1.

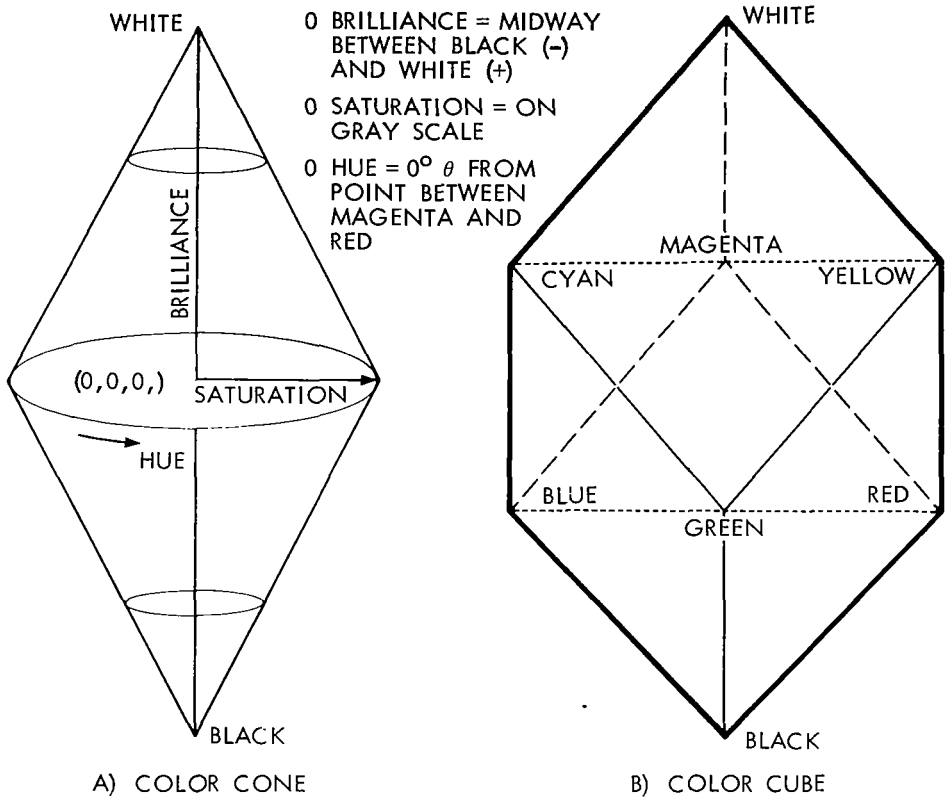


Figure 8. Color Mode

Since the DX-1 equipment is designed for specification of color in sixteen intensities, for each of the three primary colors red, green, and blue, an optional mode of specification corresponding to this scheme will also be provided by operator CMODE 2 (Figure 9).

The method for conversion from color cube values to color cone values will assume that the hexagonal cross section which is located normal to the internal black-white diagonal, and halfway between these extremes, corresponds directly to the maximum radius circular cross section of the color cone. A more correct mapping would relate this circle to the edge path of the cube which

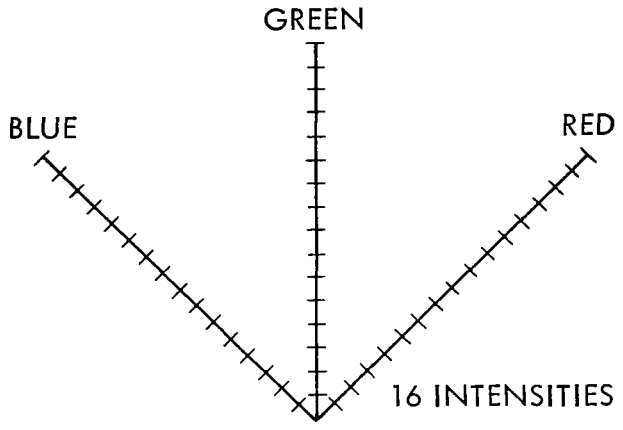


Figure 9. Color Mode 2

connects the primary and secondary color corners. Either alternative may be selected on a rational basis. The hexagonal cross section contains points of equal brilliance, which is one attribute of the circular cross section. The non-planar surface through the cube edges contains the pure or fully saturated colors, red, blue, and green, another attribute of the circular cross section. Since the hexagonal cross section alternative appears to be a simpler mapping problem, it will be used initially at least in CMODE 1.

In determining the color to be ASSOCIATE'd with an object, only the last color picked will be related to the object. Thus, if the wrong color is picked, the user may immediately pick a second which, after choosing an object and picking the ASSOCIATE button, will be assigned to that object.

This sub-mode can return to whatever mode called it by picking the RETURN button.

LABM

This changes mode to label sub-mode where the axes are suitably labeled alongside the tick marks.

Now, the graph is complete. The user can now manipulate, modify, save, or erase the graph. An example of a possible sequence of user actions is shown in Figure 10.

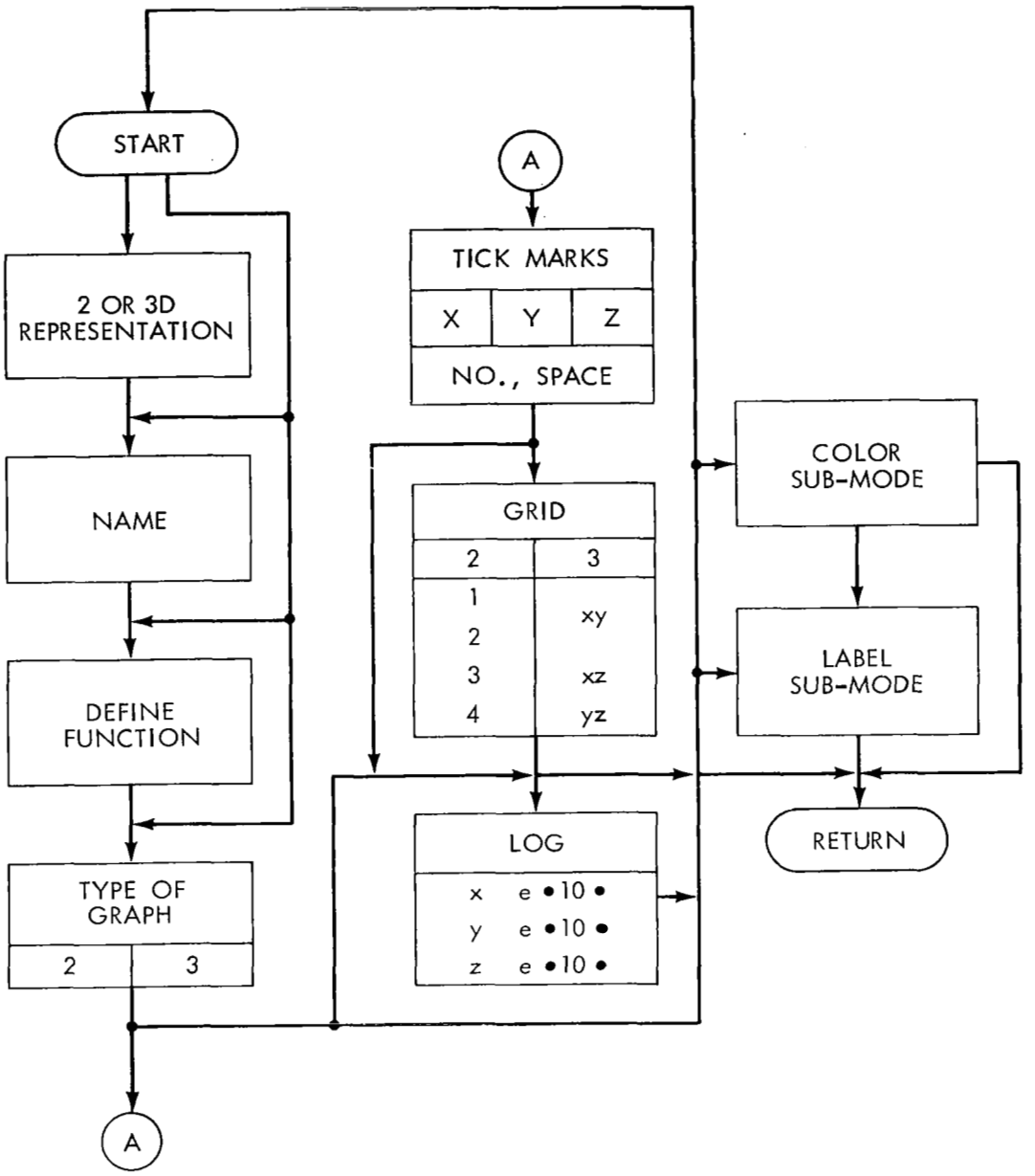


Figure 10. A Possible Set of Operator Sequences

3.2 DISPLAY MANIPULATION

Display manipulation embodies those actions the user takes in pointing at and drawing on the CRT, whose end results are to change the shape and/or position of a displayed item or set of displayed items. This is quite different from actions performed by the user which explicitly supply parameters to subroutines. The DEFF mode discussed in the previous section

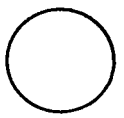
is a prime example of explicit input of parameters to a subroutine. The range values supplied by the user are needed by data scanning and scaling routines before the graph can be presented. In many cases the user does not know his data ranges precisely. For other actions, such as moving a display, changing its size or specifying data ranges, it is certainly possible to have modes which the user requests which require "fill ins." The use of "pick" type of "menu" operators is appropriate for certain types of applications but consideration should be given to making use of more recently developed techniques. Picking becomes very inflexible because it requires that the user constantly switch modes to perform his appropriate actions. Instead, we propose the incorporation of a symbol recognizer which, when interfaced with the proper manipulation request analyzer, would allow the user to perform many manipulation tasks by a more implicit means. The proper marriage of a symbol recognizer and action analyzer can provide for a fluent conversation between the man and the machine.

3.2.1 Symbol recognizer. Although it is not appropriate to present a detailed discussion of symbol or character recognizers[14,15,16,17] here, a brief discussion will define their requirements for the intended use. A symbol recognizer is a program which will monitor the state (i.e., x-position, y-position, pen on-off switch) of a stylus-like input device (i.e., Rand tablet[18] or Sylvania Data Tablet[19]), and at the proper time output the appropriate code and related parameters describing the recognized symbol. These additional parameters are usually some description of character size and position. Some recognizer dictionaries are based on statistically averaged patterns while others are trainable and become very adaptive[20] to an individual user's style of drawing symbols. In either case, the recognizer chosen will be sufficient if it can recognize the display manipulation symbols discussed below.

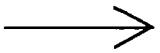
3.2.2 Display manipulation symbols. The symbology* presented below is not necessarily complete, but gives examples of the type of

*Although the symbols shown may not be to a particular individual's liking, they were chosen with the notion that they symbolically relate to the function performed. With a trainable recognizer however, it would be no great task to let the user redraw the manipulation symbols to fit his own graphic needs.

operators one would wish to use to interact with and manipulate a data display. Their individual descriptions cannot be complete without an understanding of the manipulation request analyzer discussed in Section 3.2.3.



Identify - The item circled is to be identified and used as an operand by the next operator. Identifiable items would be such things as data points, origin of axes and end points of axes.



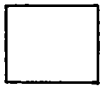
Move - Move the identified item to the point specified by the arrow head. (Arrows may be of any length and sense.) Since certain items of the display (i.e., origin of axes, end points of axes) relate directly to the data displayed, operations performed at the recognition of this symbol must be performed on both the identified item and those items related to it.

Save Valid - Items (i.e., data points) previously identified are to be saved as valid data and can be used later as input to analysis procedures.

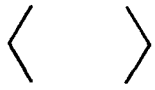
Save Invalid - Items (i.e., data points) previously identified are to be saved in an anomaly file and can be analyzed later.

Show Value - Display the exact value of the identified data point.

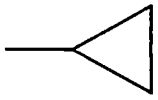
Scrub - Erase the underlying item from the screen.



Window - Change the size of the complete display to fit inside the drawn box and place it there.



Range - Accept a closed curve as a data space for enclosing wanted data. All those points falling inside the data space would be identified for further action.



Display - Display the previously identified items.

3.2.3 Manipulation request recognizer. The task of this program is to monitor the symbols outputted by the symbol recognizer and initiate the appropriate procedures. In some cases the appropriate procedure will be a further call to the recognizer program to request further symbols; in others, it will be requests for data storage, display regeneration or data base interaction. The manipulation recognizer itself may be thought of as being syntax* directed, and is based in part on notions presented in Anderson's thesis.[21]

4.0 LANGUAGE SURVEY

Without attempting to discuss the final machine configuration in which the data analysis system will exist, criteria can be stated which seem necessary to guide the choice of an appropriate system design language (SDL) for the user interface.

The goal of this portion of the research is to find an SDL which will allow the building of the appropriate sections of the user system as envisioned by Chapter 4 of the previous report.[1] Although the present effort concentrates upon examining those aspects which deal directly with specification of graphical representations, some thought must be given to the data analysis computation which is necessary to support the displays. The exact steps needed to give the user this support will not be discussed here, since they obviously are related to a discussion on display primitives.

The system design language is the implementation tool for a skeletal system for pilot testing of the display representations. Since the object of the on-line system is to put the full power of the console at the user's fingertips, the user language designer must examine the SDL capabilities carefully.

The important criteria are: ability to specify data structures and processing of them; the ability to do analysis of inputs from the user's console; availability of I/O commands to support displays; the ability to easily specify and modify procedures; and mathematical and algebraic processing capabilities. It is obvious that systems having all these capabilities could be written in assembly language, but it seems appropriate to look for a more powerful language for the planned implementation. To our knowledge

*Although no syntax of pictures exists as such, a syntax for use with the manipulation symbols can be generated.

there exists no language which seems to readily meet these specifications. At best, the ones that do exist are oriented to some of these criteria and seem lacking in the others. Some researchers have designed their own SDL's by using compiler-compilers, which unfortunately lead to languages which become very configuration dependent. The present study examines the availability of those languages which meet at least part of the criteria and discusses tradeoff considerations for final selection of an SDL. Most SDL candidates seem oriented toward list and string processing because this feature seems very important for analysis of the language.

In this sequel we will discuss languages for processing linked data sets. The motivation behind this is twofold. The man-machine interaction can be thought of as a conversation in some language, the elements of which are the primitives which can be thought of as a user vocabulary. The system must be able to analyze these statements. The list and string processing languages lend themselves nicely to the algorithms needed for this type of analysis. The second important feature of these languages is that they allow for dynamic storage allocation and data structuring. Data structuring is particularly important in graphic communication, not only because it retains the physical structure of pictures, but also because it can be used to represent abstract relationships between data elements.

Guidelines for choosing such a language exist to some extent as a report[5] of a Subcommittee of the ACM Special Interest Committee on Symbolic and Algebraic Manipulation. To complement this report, members of the group studied those languages which received strong recommendations and evaluated them based on the present system requirements. Those languages of interest are discussed below.

L6[6], one of the most widely known of the linked-block languages, is very appropriate for the data structuring and handling which must take place in the data analysis system. Since the system programmer can define small cell lists for his language structure, define large blocks for his data storage, and write the appropriate macro routines to process these structures, a coherency of data storage is maintained. This coherency allows for a completeness which is logically more advantageous than using one programming language to analyze the user language string and another to manipulate the data structure. The CORAL language which has been used extensively by Lincoln Laboratories for graphics work is similar to L6 but must be rejected since it is inaccessible, except on the TX-2 computer. L6 also becomes unapproachable because there are no implementations of it for the PDP-1.

Another candidate which was considered is SLIP.[8] Because SLIP is embedded in FORTRAN, it provides list processing capabilities in a numeric algebraic language. However, the availability of FORTRAN for the PDP-1 is essentially non-existent and SLIP had to be disregarded.

Of obvious interest for the future are LISP2 and FORMULA ALGOL. Both these languages are much more powerful than their predecessors, but cannot be evaluated properly until they leave the experimental stage and have been implemented and tested for a period of time.

The skeletal system for testing the display primitives was implemented on a modified version of LISP 1.5[9] for the PDP-1, since LISP comes closest to providing the capabilities needed for the language processing. Although LISP does not inherently contain the facilities for defining large data blocks, its list properties and linkage with machine language routines will allow adequate testing of the display primitives and representations. It is recommended, however, that serious considerations be given to considering L6 or a similar language for final implementation of these portions of the present study which prove useful.

5.0 DATA STRUCTURES

A number of data structures were analyzed to determine what would be a suitable data structure for scientific analysis of experimental data. These included the CORAL data structure[7] which SKETCHPAD used; the data structure for AEDNET,[10] a simulator for non-linear electronic networks; the data structure for Pictorial Encoding Language (PENCIL);[11] a proposed data structure for hierarchical graphical entities[12] for the DX-1 Experimental Processor; and data structure for list processing languages such as LISP.[9, 13]

The data structures for the list processing languages are inefficient for data displays primarily due to their high ratio of pointers to words of information. Display information is not as amorphous as the textual information expected for list processing languages, and the use of a large number of pointers and small blocks of information is expensive in both core storage and computing time.

The data structure for AEDNET used ring-associated blocks. But the linking of the blocks and the structure of the blocks are too closely oriented to the display of electronic networks to be used for scientific data analysis.

The CORAL data structure, the PENCIL data structure, and the proposed data structure for the DX-1 are all general data structures but were primarily evolved for design type displays as opposed to data displays.

The CORAL data structure consists of blocks of varying length which are used to represent items or entities. The blocks are tied together in rings by means of pointers. The rings represent the associations of the blocks.

The PENCIL data structure is similar to the CORAL data structure but all entities are ultimately composed of points and lines which are combined into "masters" and "sub-picture masters" to generate the display.

The proposed data structure for the DX-1 consists of variable length blocks for various entity types tied together in a ring. At present the proposed data structure is only capable of displaying two-dimensional figures.

The latter three data structures were designed so that the user could add, delete, modify, and build complex entities. The sophisticated ring design was developed to facilitate these operations. Although this is desirable for design work, scientific analysis of experimental data would not have a high priority for these operations. The bookkeeping time saved by utilizing a simpler data structure would be significant and could be used for numerical analysis of the experimental data.

Also due to the variable size blocks available in the above data structures, garbage collections are lengthy and time consuming. In the analysis of large quantities of data, garbage collections would be more frequent and therefore more costly.

A simple ring associated data structure that allows for efficient and flexible use of core storage and which would need only limited garbage collections is recommended. A ring association will be necessary since the user will be manipulating the data and will probably wish to construct some simple entities to aid in his data analysis. Also the user will be interested in grouping, adding, deleting, or referencing his data.

In trying to develop a data structure an attempt was made to be machine independent and general enough to handle a large class of displayable objects and the manipulation of these objects. As a result, we devised a data structure which we feel will give us a prototype for pilot testing purposes. Our data structure is the organization of the Data Display Base and the Current Display Base of the display configuration represented in Figure 11. The scope coordinate base and the display commands are much too machine oriented to be considered here. The display configuration in Figure 11 would allow quick manipulation of data by eliminating unnecessary transformations and allowing the manipulation to take place at the proper level.

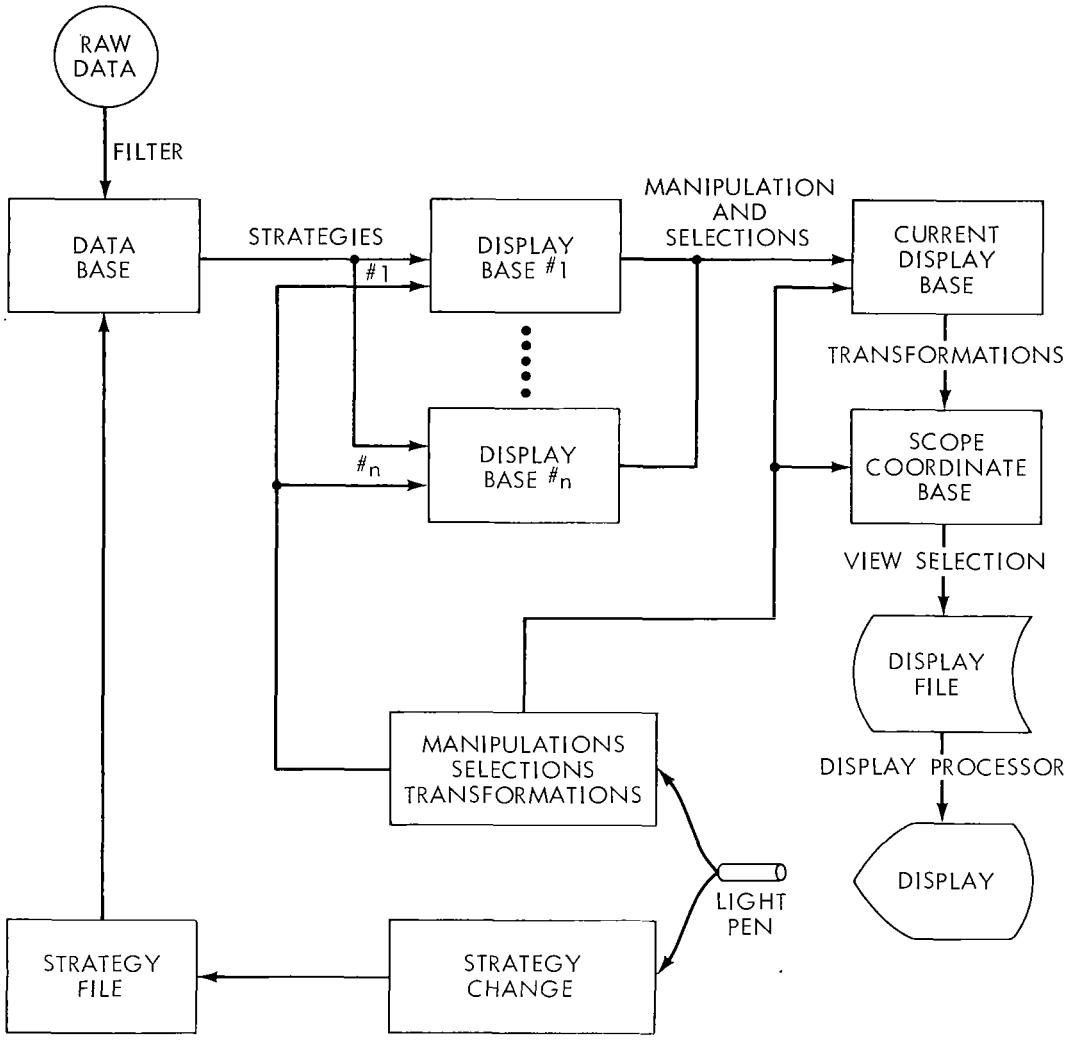


Figure 11. Advanced Display Configuration

The data structure suggested here is not necessarily in the best possible form. The machine to be used, the language used, and the data which is to be displayed may dictate significant changes in the data structure. Refinements and extensions will almost certainly be wanted.

The data structure can be viewed as a two-level ring (Figure 12). The "upper" ring is a ring of headers. Headers identify individual displayed objects which can be manipulated by the user. The "appended" rings are the entities. Entities define the shape and orientation of the object. The Scope Coordinate Base and the Display Commands of Figure 11 are derived from the information contained in the entities. No entity may be referenced by the user except through its header and no header will have

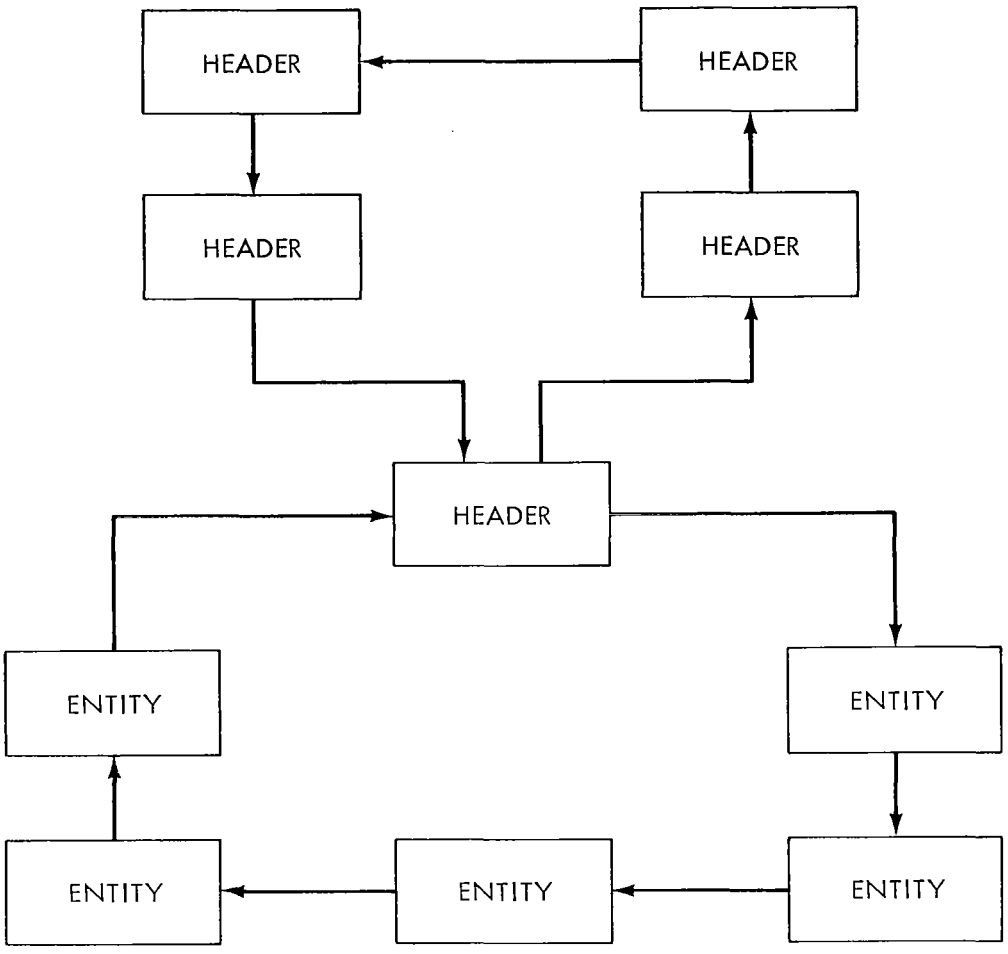


Figure 12. Two-Level Ring

another header as an entity. The data structure is somewhat limited in the construction of complicated objects by these restrictions, but not severely. We expect that processing time saved by not having to travel through a tree type structure will allow more of a real-time response to the user. Processing time is at a premium especially for kinetic displays.

Free storage is separated into three types: the free header ring, the free entity ring, and blank or zero core. Zero core is the part of the allocated data base area that has not yet been used by the display data base. This eliminates the need to initialize free core into a ring. The free header ring is a ring of header blocks no longer used by the display data base. The header blocks are placed into the free header ring as they are released. The free entity ring is similar.

Whenever an entity or header block is needed, it is first released from the free rings. If there are no blocks left in the free string, then a block is generated from zero core. If a header block were needed and both the free header ring and the zero core were exhausted, a free entity block would be released and converted to a header block. If an entity block were needed and both the free entity ring and zero core were exhausted, a search would be made for two free headers in consecutive core locations which would be released as an entity block. If no blocks are available for new headers or entities, an appropriate message would be given to the user. No real garbage collection is necessary and only in extreme circumstances is an extensive search for free storage necessary.

The display would be generated by having a processor travel around the active header ring. At each header the processor would loop through the associated entity ring (see Figure 12). An examination of the entity code would tell the processor whether the entity was a point, line, circle, text or otherwise. Depending on the type of entity, the processor would transfer control to an evaluator which would generate scope coordinates or possibly display commands from the information contained in the entity blocks. For example; two entity blocks, one containing the center point, the other containing either the radius or some point on the circumference, would supply enough information to generate a circle.

The data can be manipulated but only at the header level. The exceptions to this would be for the addition or deletion of some part of an object. This would be done at the entity level. Each object represented by a header block has an object code. Duplicates of an object would be the same figure after having undergone a rigid transformation such as translation or rotation. These duplicates would contain within their header blocks identical object codes to the original. In this way, changes made to any of the objects with the same code could be extended to all similar objects. If the change is not to be made to all similar objects, then the changed object would receive a new object code.

Two or more objects could be combined into one object by tying the entity codes together and using one header block where two were needed. No object will be easy to break down into its component parts after a combination. (This may not be possible at all, and certainly won't be encouraged.)

Display transformations such as translations, rotations, and projections would be done by applying a matrix multiplication to the vectors describing the entities. These transformations will be applied to all entities belonging to a given header.

The header block consists of four words as shown in Figure 13. Two header blocks reside within a larger block called a double header block. If a header is needed from zero core, a double header block is released. One of the header blocks is assigned as needed to the active header ring. The other header block is assigned to the free header string. This double header block enables the free entity string to "steal" from the free header string under certain conditions without a costly garbage collection.

The header code indicates that the block is a header block and whether it is active or free. The duplication identity code enables copies to be associated. The entity pointer is an address pointing to the first entity block for this header. The backward pointer is the address of the previous header in the active (or free) header ring. The forward pointer is the address of the next header in the ring.

The entity block has eight words as shown in Figure 14. The entity code specifies the type of entity it is. The X, Y, Z and W coordinates define a vector in 4-space (needed for homogeneous coordinates). These words could be used to contain other data for other entity types. The color and form of the object is defined in the next word; the form indicates a type of line; broken, dotted or otherwise. This word could also contain other data depending upon entity type. The back to header pointer contains the address of the first word of the header block. The forward pointer contains the address of the next entity block, or if this is the last entity block it will contain the address of the entity pointer in the header block.

Text or functions to be evaluated could be represented within our data structure by defining an entity type for them and then allowing words 1-5 of the entity block to contain the textual or function data.

Before this data structure can be implemented it will be necessary to define the entity codes and construct the evaluation routines. Bookkeeping for the free strings and the transformation of the data structure into scope coordinates and display commands will have to be set up. These functions are too machine and problem oriented to define at the present.

A hypothetical problem is presented to illustrate the use of the data structure (see Figures 15-17). The pointers are abbreviated as follows:

ARP	Active Ring Pointer
FHP	Free Header Ring Pointer
FEP	Free Entity Ring Pointer
ZCP	Zero Core Pointer
NOSN	Next Object Serial Number

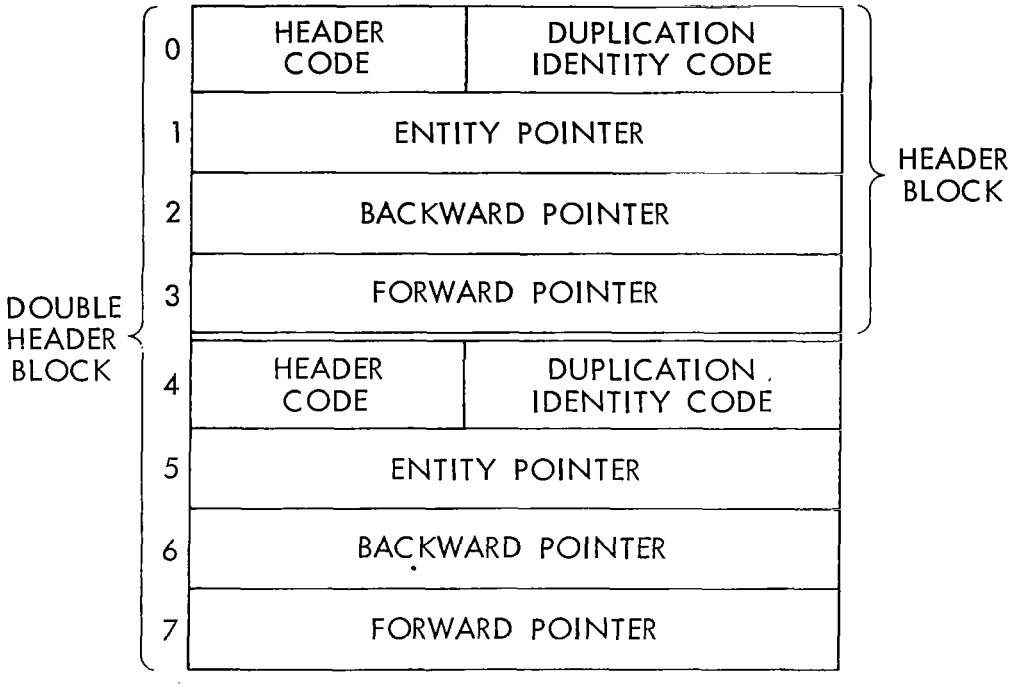


Figure 13. Header Block Structure

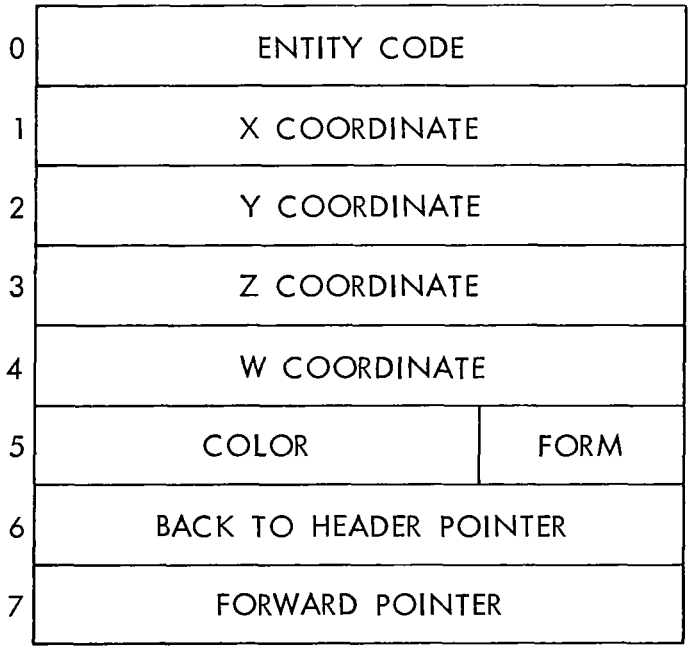
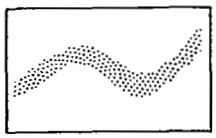


Figure 14. Entity Block Structure

USER ACTION: DISPLAY A SET OF DATA POINTS

SCOPE:



POINTERS: ARP = 1000
FHP = 1004
FEP = 0
ZCP = 1808
NOSN = 02

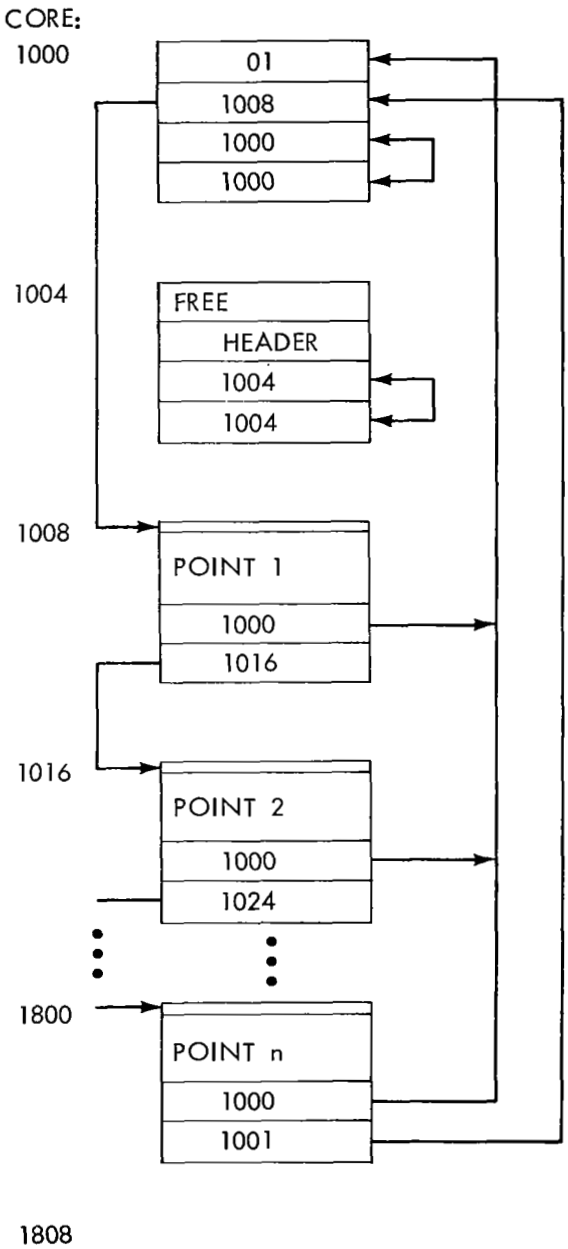


Figure 15. Data Structure Example 1

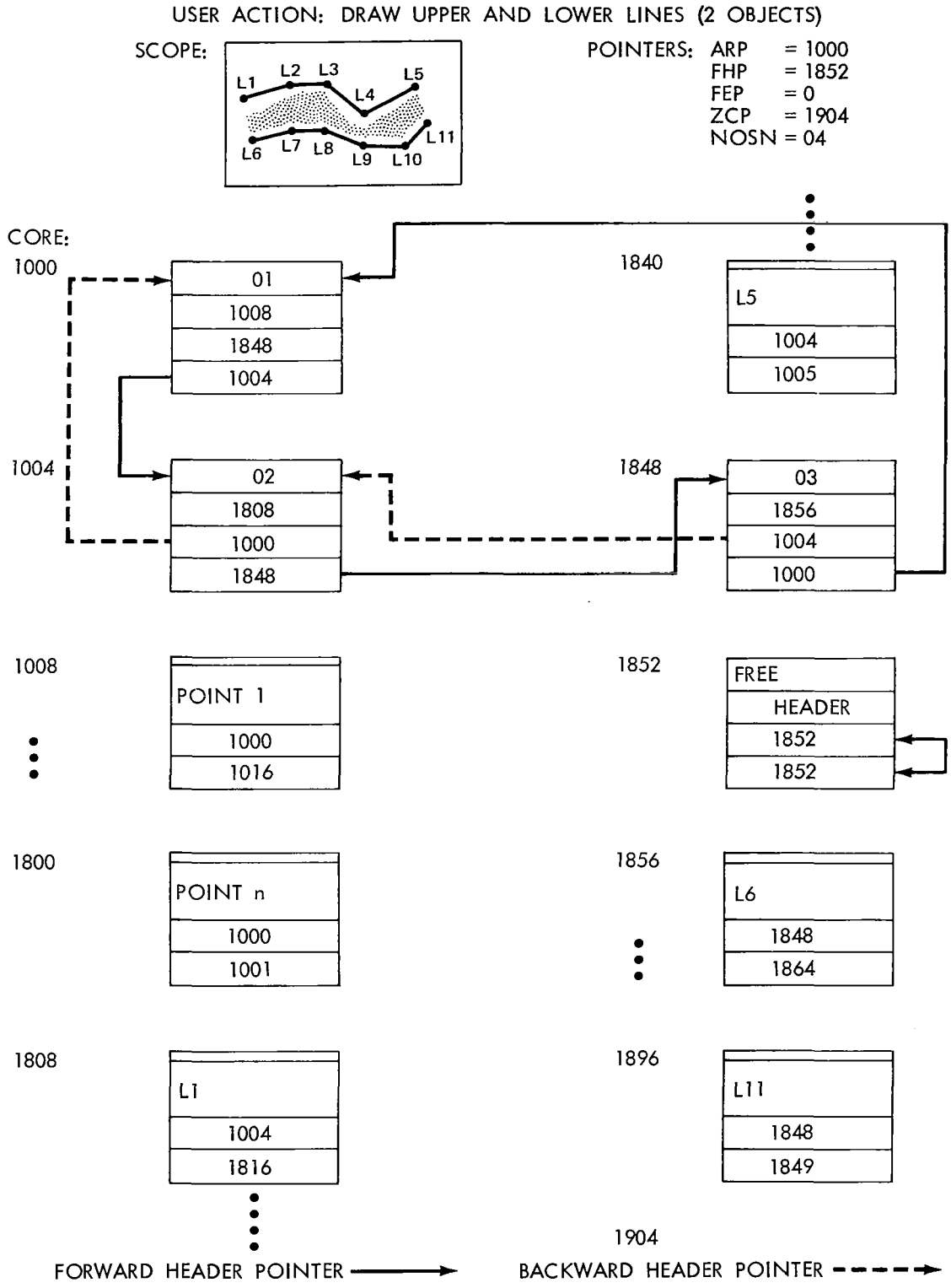


Figure 16. Data Structure Example 2

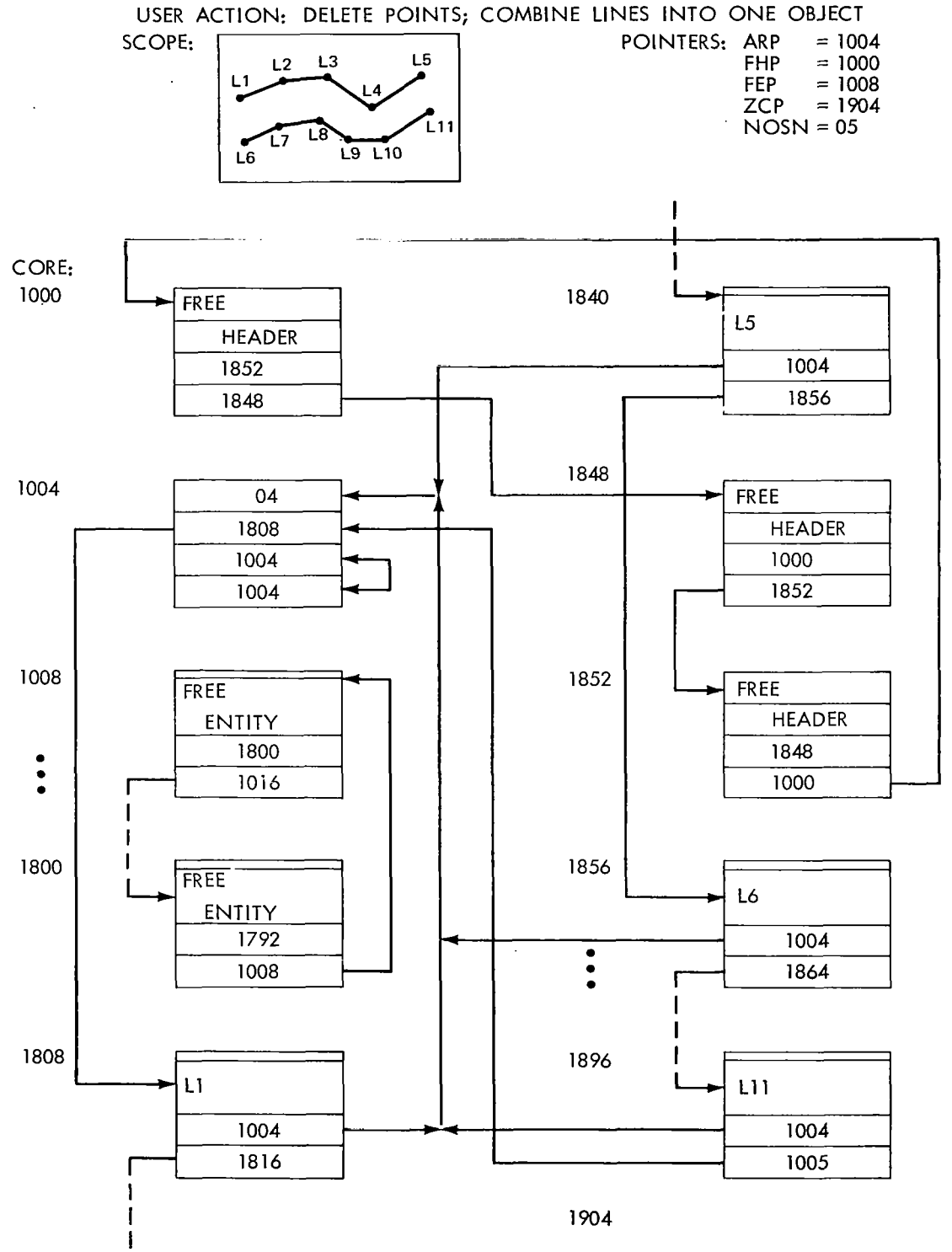


Figure 17. Data Structure Example 3

6.0 A FORMAL SPECIFICATION OF A USER ORIENTED DATA DISPLAY LANGUAGE

6.1 PHILOSOPHY OF A USER ORIENTED DATA DISPLAY LANGUAGE

A User Oriented Data Display Language (UODDL) is the language through which a user can specify the format of a graphical display of his data. The user's specifications or input can be entered from a drawing device such as a Rand Tablet, a light pen, typewriter, or other interactive devices. The syntax rules which specify the UODDL not only are the rules the computer uses to decode input strings, but also are a description of the user's grammar. They specify how the format of a graphical display is affected by the user's actions. Some unconventional terminology has been introduced into the UODDL syntax rules for this purpose.

The user specifies the parameters for a set of pre-packaged modular display programs through the UODDL. Since these programs are executed at the time of display, the specification of parameters need not be strictly ordered. Also the parameters can be reset at any time before execution. These qualities permit flexibility and ease of use.

The user's input can be regarded as a string, ordered to some degree by time. Each action of the user can be interpreted as a symbol or a string of symbols. For example, at time t_0 the user performs some action, α; at time t_1, β; and at time t_2, γ; where t_0, t_1, t_2 are sequential. He has then created the string $\alpha\ \beta\ \gamma$. Now suppose the action, γ, is better represented as the string, $\delta\ \epsilon$. Then the user has created the string $\alpha\ \beta\ \delta\ \epsilon$. An example of a string such as $\delta\ \epsilon$ would be the result of the user's action being analyzed by a two-dimensional syntax recognizer. The two-dimensional syntax recognizer would restructure the action, γ, according to position and orientation into the linear string, $\delta\ \epsilon$: Then position and orientation could be considered secondary string ordering.

Each time a symbol or string of symbols is added to the user's input string, a program called the User Action Analyzer (UAA) would examine the string to determine its legality with respect to the UODDL syntax rules. If the concatenation of a new symbol to the input string makes a previously legal input string illegal, the new symbol is considered a delimiter and the start of a new string. The inputted string minus the new symbol is analyzed for possible reconstruction and action by the UAA according to syntax rules. It is also possible that the UAA will reconstruct the input string before it is completed. Thus, ordering of the input string is not strictly according to time or position.

Two important concepts in the specification of syntax rules for the UODDL are independent strings and additive independent strings. These strings are the parameters or lists of parameters for the display programs. When a string which satisfies the syntax rules for an independent string has been formed, the value of the independent string is reset to the currently inputted string and the previous value is lost. All independent strings have initial values which are specified in the syntax rules.

The additive independent string is different from the independent string in that a substring of a type specified by the syntax rules can be formed at any time. This substring is then concatenated to the current value of the additive independent string creating a new value. In this way lists whose substrings are defined in widely separated actions can be constructed. To remove a substring from an additive independent string, a formal deletion is necessary.

Another important property of the independent and additive independent strings is the fact that they need not be redefined each time the user wishes to construct a string, satisfying a syntax rule which includes independent or additive independent strings. If the user does construct a string which would satisfy a syntax rule except for missing independent or additive independent strings, then the current values of the missing strings are inserted, and the syntax rule is completely satisfied. This allows the user to use a parameter many times with only one real specification of it. A good example is the specification of minimum and maximum values for the four possible axes (x, y, z, and color). If the minimum and maximum are to be the same, the user specifies them once, and then merely assigns these values to each of the four axes by creating a string of actions according to the syntax rules with the exception of respecifying the minimum or maximum, e.g., assume the minimum or maximum are preset, for the x-axis the user merely creates a string with the single symbol X, the action specifying the x-axis. The current maximum and minimum are automatically inserted.

The UAA will have to keep track of string levels, since independent and additive independent strings are used in the syntax definition of other independent or additive independent routines. Then if the user has specified the value of an independent or additive independent string within the specification of another independent or additive independent string, the UAA will not terminate the specification of both strings when only the inner string is to be terminated.

The syntax rules for the UODDL have been designed to avoid ambiguities. In a system where the user has a full range of resetting independent strings,

ambiguities cannot easily be resolved by context, if at all. However, by appropriate use of symbols such as comma most ambiguities can be easily avoided. Illegal strings also require a special thought. The UAA should recognize a string which does not fit any syntax rule as illegal, and should delete it from the input string. This deletion should leave the user's string as intact as possible. An example of this would be the appearance of a letter in the middle of a string specifying a number, e.g., 12a34. Here the letter would be removed and the number (1234) would be as if no letter has appeared. This type of resolution is not always possible. If the user makes a string illegal by an action which would initiate a new string, then the UAA must assume the action was correct and act accordingly.

In the syntax rules for the UODDL, color is used in a number of ways. It can be used for improving the graph by coloring the axes, labels, and grid lines or tick marks. Also it can be used to identify a point as belonging to a certain data set, or as having a certain value in an extra dimension. In displaying the data according to the parameters specified by the user, the data set color is superseded by a color assigned by either the color axis or data color scheme.

6.2 DEFINITION OF TERMINOLOGY

6.2.1 Special UODDL terminology

Definition: A string is an arbitrary number of concatenated symbols which are members of a set called a terminal set, usually represented by T.

Example: Some strings of the terminal set T = {a, b, c} are a; aaa; abc; bab; ccabccbaabbc.

Definition: The empty string, λ, is the string with no symbols.

Example: a = abc; $\lambda a = a\lambda$ = abc

Definition: A type or form of a string is the set of strings satisfying a given syntax rule and is represented by a name enclosed in angle brackets, (i.e., <name>)

Example: T = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,}
<digit> :: = 0|1|2|3|4|5|6|7|8|9
<number> :: = <digit> | <number> <digit>
<number> and <digit> are types of strings.

<digit> and <number> <digit> are types of strings of the form <number>.
0 is a string whose form is both <digit> and <number>.
736 is a string whose form is <number>.

Definition: An independent string is a string with the property that the User Action Analyzer (UAA) only retains one string of that form in core memory at any one time. It is available for insertion between two symbols according to syntax rules. An independent string is enclosed in double angle brackets (i.e., ≪name≫).

Note: Whenever the UAA detects a user imputted string in the form of an independent string, the value of the independent string is set to the new string and the old value is lost.

Example: ≪item 1≫ :: = <thing 1> | <thing 2> <thing 3>
≪item 2≫ :: = <thing 4> ≪item 1≫ <thing 5>
Suppose ≪item 1≫ = <thing 1>. Then the user constructs the string <thing 4> <thing 5>. The UAA should interpret this as the syntax rule for ≪item 2≫ missing only the predefined ≪item 1≫. Therefore, the value of ≪item 2≫ is immediately set to <thing 4> <thing 1> <thing 5>.

Now if the user constructs a string of the form <thing 4> <thing 2> <thing 3> <thing 5>, the UAA should recognize <thing 2> <thing 3> as a legitimate form of ≪item 1≫. Then the value of ≪item 1≫ is set to <thing 2> <thing 3> and the value of ≪item 2≫ is set to <thing 4> <thing 2> <thing 3> <thing 5>.

Definition: The initial value of an independent string is the setting of a form to a certain string so that even if such a form has not been defined by the user, it can be assumed to have a value. Initial value is represented by I [a] where a is a given string.

Example: ≪x≫ ::= I[ab] | <something>
<y> ::= 0 ≪x≫ 1
If the user constructs a string of the form 01, not having previously defined ≪x≫, the UAA will interpret 01 as 0ab1 since ≪x≫ has the initial value ab.

Definition: The additive independent string is a string with the properties of an independent string except that it can be added to in widely separated strings. It is denoted by a name enclosed in triple angle brackets.

Example: <<<x>>> ::= abc | <<<x>>> abc | I[λ]
The user first constructs the string abc changing <<<x>>> from an empty string to a string having the value abc. At a later time, after defining other strings, the user constructs the string abc. The UAA recognizes this as an attachment to <<<x>>> and resets <<<x>>> to the new value of abcabc.

Definition: A one level exhaustive string with respect to <y> is a string of the form <a><y> <b> . . . <a> <y> <b> where each type of string used in the syntactic definition of <y> appears exactly once. It is represented by:

P [<a> <y> ! <b>]

Example: <a> ::= a|b
<b> ::= e|f
<y> ::= 1|0
P [<a> <y> ! <b>] represents strings such as a0eale; blea0f; alea0e; b0ealf; alfa0f. But a0ea0e, b1da1e; a0e; b0f; a1f are not represented by
P [<a> <y> ! <b>]

Example: <x> ::= a|b
<y> ::= <s> | <t>
<z> ::= n|m
P [0 <x> !] represents 0a0b or 0b0a.
P [<x> <y> !] represents a <s> a <t>;
a <t> a <s>; a <s> b <t>; b <t> a <s>;
a <t> b <s>; b <s> b <t>; b <t> b <s>.

P [<x> <y> ! <z>] represents strings such as
a <s> n a <t> m; b <t> mb <s> n;
b <s> ma <t> m.

Definition: Construction of a new string through replacement is defined as the substitution of one string for another and is represented by: R [<s> ; <t>].
The UAA upon recognizing a string of the form <s> immediately replaces it with a string of the form <t>.

Example:
<y> :: = a | b
<z> :: = 0 | 1
<w> :: = k | 1
<x> :: = R [<y> <z> <w> ; <y> <w>]
Suppose the user constructs the string a0k, the UAA will recognize this and change the string to ak. Then ak is a string satisfying the syntax rule of <x>.

Definition: The construction of a string by exclusive addition is defined as the deletion of a substring of the form $<y_1>$ <x> $<y_2>$ from a string and concatenating the result with another string of the same form but where the incoming substring of the form <x> is required of the form <x>. The incoming substrings $<y_1>$ and $<y_2>$ need only be of the same form as the outgoing substrings.
Exclusive addition is represented by:

$$A\ [a\ ;\ <y_1>\ <\underline{x}>\ <y_2>]\ ,$$

The underlined portion is the portion that must match exactly.

Example:
<<<x>>> :: = I [a 0 b 1] | A [<<<x>>> ;
<letter> <number>]
<letter> :: = a | b | c . . . x | y | z
<number> :: = 0 | 1 | 2 . . . 8 | 9
The user constructs the string c3, the UAA recognizes c3 as a valid attachment to <<<x>>>, finds no previous substring of the form <letter> 3, and sets <<<x>>> to a0b1c3.

Later the user constructs the string d1. The UAA recognizes d1 as a valid attachment to <<<x>>>, finds substring b1, deletes it from <<<x>>>, and resets <<<x>>> to a0c3d1.

Definition: An additive independent string can be reconstructed through the operation of deletion. A specified string followed by the symbol del will cause a substring of a form to be deleted if a portion of the substring matches the specified string. Deletion is represented by:

D [<y> del; <<<x>>>; <w> <y> <x>]

Example:
<y> ::= a|b
<w> ::= k|1
<z> ::= p|q
<<<x>>> ::= D [<y> del; <<<x>>>; <w> <y> <z>]
Suppose <<<x>>> currently has the value of kapkbq. The user constructs the string b del. The UAA will interpret this string as a deletion with b matching the b in the substring kbq and will set <<<x>>> to the value kap.

6.3 TERMINAL SET

A terminal set, T, sufficient for the proposed syntax rules is:

T = {a, b, c, . . ., x, y, z, blank, comma, 0, 1, 2, . . ., 8, 9, min, max, neg, num, x, y, z, linear, log, exp, power, col, red, yellow, green, cyan, blue, magenta, white, black, brill, sat, hue, grid, tick, nogridtick, #, sp, line, seg, dot, contour, surface, axis, label, gridtick, lambda, nam, del, data, sav, dis, DS1, . . ., DSN}

The underlined items represent symbols, abbreviations, or the words themselves which are considered units of the users input string. The definitions of the underlined items are:

blank	an alphanumeric blank
comma	an alphanumeric comma
min	a minimum

max	a maximum
neg	numeric negative sign
num	number reset
x	x-axis
y	y-axis
z	z-axis
linear	linear scale
log	log scale
exp	exponential scale
power	power scale (i.e., x^2, x^3,. . .,x^n)
col	color mode
red	red
yellow	yellow
green	green
cyan	cyan
blue	blue
magenta	magenta
white	white
black	black
brill	brilliance
sat	saturation
hue	hue
grid	grid lines
tick	tick marks
nogridtick	neither grid lines nor tick marks
#	grid lines or tick marks will be spaced n to an axis
sp	grid lines or tick marks will be spaced at every nth value on axis
line	data will be represented as line or bar graph

seg	data will be represented as connected line segments
dot	data will be represented as dots
contour	data will be represented as contoured data with z axis perpendicular to screen
surface	data will be represented as a surface constructed by othogonal surface lines
axis	graph axis lines
label	graph labels or label mode
gridtick	grid lines or tick marks on displayed graph
lambda	empty string; null value
nam	name mode
del	delete mode
data	data mode
sav	save mode
dis	display mode
DS1 ⋮ DSN	sets of data

6.4 SYNTAX RULES

```
≪name≫ :: = nam <letter> | ≪name≫ <char> |
            I [λ] | R [nam lambda; λ]
<char> :: = <letter> | <numeral> | comma
<letter> :: = a|b|c . . . |x|y|z|blank
<numeral> :: = 0|1|2 . . .|8|9

≪min≫ :: = min|min <number> | I [min]
≪max≫ :: = max|max <number> | I [max]
<number> :: = <numeral> | neg <numeral> | <number>
               <numeral> | R [<number> neg; neg
               <number>] | R [neg neg <number>;
               <number>]| R [<number> num; 0]

≪x limits≫ :: = x ≪min≫ ≪max≫ | x ≪max≫ ≪min≫ |
               I [x min max]
```

≪y limits≫ :: = y ≪min≫ ≪max≫ | y ≪max≫ ≪min≫ | I [y min max]
≪z limits≫ :: = z ≪min≫ ≪max≫ | z ≪max≫ ≪min≫ | I [z min max]

≪x scale≫ :: = x <scale> | I [x linear]
≪y scale≫ :: = y <scale> | I [y linear]
≪z scale≫ :: = z <scale> | I [z linear]

<scale> :: = linear | log <base> | exp <base> | power <base>
<base> :: = e | <number>

≪color axis≫ :: = col <min color> comma <max color> | I [col black comma white]
<min color> :: = ≪color≫
<max color> :: = ≪color≫

≪color limits≫ :: = col ≪min≫ ≪max≫ | col ≪max≫ ≪min≫ | I [col min max]

≪color scale≫ :: = col <scale> | I [col linear]
≪x gridtick≫ :: = x <gridtick ind> ≪spacing≫ | I [x nogridtick # 10]
≪y gridtick≫ :: = y <gridtick ind> ≪spacing≫ | I [y nogridtick # 10]
≪z gridtick≫ :: = z <gridtick ind> ≪spacing≫ | I [z nogridtick # 10]
<grid tick ind> :: = grid | tick | nogridtick
≪spacing≫ :: = # <number> | sp <number> | I [# 10]

≪type≫ :: = line | seg | dot | contour | surface | I [dot]

≪color≫ :: = red | yellow | green | cyan | blue | magenta | white | black | <color cube selection> | <color cone selection> | I [white]

<color cube selection> :: = <prime color> <intensity> | A [<color cube selection>; <prime color> <intensity>]
<intensity> :: = 0 | 1 | 2 | . . . | 13 | 14 | 15
<prime color> :: = red | blue | green
<color cone selection> :: = <color cone coord> <number> | A [<color cone selection>; <color cone coord> <number>]
<color cone cord> :: = brill | sat | hue
<<<graph part colors>>> :: = I P [white <graph part>!] | A [<<<graph part colors>>>; ≪color≫ <graph part>]

<graph part> :: = axis | label | gridtick
≪graph label≫ :: = nam <label> | R [nam label lambda; λ] | I [λ]
≪x label≫ :: = x <label> | R [x label lambda; λ] | I [λ]

≪y label≫ :: = y <label> | R [y label lambda; λ] | I [λ]
≪z label≫ :: = z <label> | R [z label lambda; λ] | I [λ]
<label> :: = label <char> | <label> <char>

<<<data color scheme>>> :: = col <relation> ≪color≫ | A [<<<data color scheme>>>; col <relation> ≪color≫] | D [col <relation> del; <<<data color scheme>>>; col <relation> ≪color≫] | R [col comma lambda; λ] | I [λ]
<relation> :: = <lower limit> comma <upper limit> | comma <number>
<lower limit> :: = <number> | min
<upper limit> :: = <number> | max

<<<graph list>>> :: = ≪name≫ P [<graph specs>!] sav | A [<<<graph list>>>; ≪name≫; P [<graph specs>!] sav] | D [≪name≫ del; <<<graph list>>>; ≪name≫ P [<graph specs>!] sav]

<graph specs> :: = ≪x limits≫ | ≪y limits≫ | ≪z limits≫ | ≪x scale≫ | ≪y scale≫ | ≪z scale≫ | ≪color axis≫ | ≪color limits≫ | ≪color scale≫ | ≪x gridtick≫ | ≪y gridtick≫ | ≪z gridtick≫ | ≪type≫ | <<<graph part colors>>> | ≪graph label≫ | ≪x label≫ ≪y label≫ | ≪z label≫ | <<<data color scheme>>>

≪data≫ :: = data P [<data specs>!]<data set> | ≪data≫ P [<data specs>!] <data set> | I [λ]

<data specs> :: = ≪color≫ | ≪x coord≫ | ≪y coord≫ | ≪z cord≫ | ≪color coord≫

≪x coord≫ :: = x <number> | R [x lambda; λ] | I [λ]
≪y coord≫ :: = y <number> | R [y lambda; λ] | I [λ]
≪z coord≫ :: = z <number> | R [z lambda; λ] | I [λ]
≪color coord≫ :: = col <number> | R [col lambda: λ] | I [λ]
<data set> :: = DS1 | . . . | DSN

<display specs> :: = ≪x limits≫ | ≪y limits≫ | ≪z limits≫ | ≪x scale≫ | ≪y scale≫ | ≪z scale≫ | ≪color axis≫ | ≪color limits≫ | ≪color scale≫ | ≪x gridtick≫ | ≪y gridtick≫ | ≪z gridtick≫ | ≪type≫ | <<<graph part colors>>> | ≪graph label≫ | ≪x label≫ | ≪y label≫ | ≪z label≫ | <<<data color scheme>>> | ≪data≫

<display> :: P [<display specs>!] dis

6.5 PRESET STRINGS

A User Oriented Data Display Language (UODDL) such as we have described should allow the user to set standard and often-used options. The concept here is similar to the concept of macros for a programming language. Since user input is a string, preset strings can be concatenated to the input string and then processed in the same mode as the user's actions. Preset strings can, on activation, reset independent strings to specified values, end a partially completed string, and begin a new string.

A preset string should have a name and an expression which is a string used as a model. Also the user must be able to indicate when he is beginning to define a preset string and when he has ended. A syntax rule for the definition of a preset string is:

< preset string > :: = beg ≪name≫ < expr > end

where < expr > is any valid string as defined in the syntax rules for the UODDL. The User Action Analyzer (UAA) should recognize and eliminate any illegal substrings when end is inputted. This will save storage and the necessity of eliminating illegal substrings each time the preset string is activated. If preset string definitions are not to be imbedded in each other, beg, and end may be the same symbol. Otherwise, it is necessary that they be different.

An option to specify dummy arguments for preset strings would probably be useful. However, this would mean that the user would have to remember the number of arguments used, what they were used for, and their order for each preset string. Or some mechanism could be set up so that when he picks a preset string he is immediately informed of the dummy argument to be inputted. A syntax rule for a preset string which would include dummy arguments is:

< preset string > :: = beg ≪name≫ < arg list > < expr > end
< arg list > :: = arg < dummy list > arg
< dummy list > :: = < letter > | < dummy list > < letter >

Each letter would represent one dummy argument. Thus, in specifying a preset string, any letters which are members of the dummy list will be replaced by a substring during activation. To avoid having a letter unusable except as a dummy argument, some convention such as using any letter following a comma as itself could be adopted.

When the preset string is activated, the user will need a set of delimiters to inform the UAA that a substring to replace a dummy argument has been started and ended.

As preset strings are specified, they should be assigned to a list of preset strings. This list should be able to have preset strings deleted. A syntax rule for such a list is:

<<<preset string list>>> :: = I [λ] | A [<<<preset string list>>> ; beg ≪name≫ <string body> end] [D [beg ≪name≫ del; <<<preset string list>>> ; beg ≪name≫ <string> end]
<string body> :: = <expr> | <arg list> <expr>

Note: A preset string now can have the syntax rule:

<preset string> :: = beg ≪name≫ <string body > end

To activate a preset string, the user should merely have to specify the name and then pick a special operator. Syntactically his actions could be represented as:

<user's input string> :: = <previous input string> R [≪name≫ set; <string body>]

where the <string body> is that for the named preset string.

If the UAA is capable, it could display the name given to a preset string and interpret action on that name (such as light pen hit) as sufficient to activate the desired string. If the UAA had this capability, then a series of symbols such as PS1, . . ., PSN representing n preset strings would be added to the terminal set. A syntax rule for the activation of a preset string could be:

<user's input string> :: = <previous input string > R [<preset string name> ; <string body >]

where the < string body> is that for the named preset string.

6.6 NEW AND RECALL

A preset string which should be incorporated in the UODDL is one which will reinitialize the additive independent strings <<<data color scheme>>> and <<<graph part colors>>> and all the independent strings except

≪data≫ ≪x coord≫, ≪y coord≫, ≪z coord≫, and ≪color coord≫. By reinitializing these strings, the user only has to specify the data to obtain a standard graph as well as resetting options to describe a new graph. As a syntax rule, reinitialization can be defined by:

<user's input string> ::= R [new; nam lambda min max x y z col x linear y linear z linear col linear col black, white # 10 x nogridtick y nogridtick z nogridtick dot white axis label gridtick nam label lambda x label lambda y label lambda z label lambda col comma lambda]

where the UAA would recognize the following strings with the indicated results:

nam lambda resets ≪name≫ to λ
min resets ≪min≫ to minimum data value
max resets ≪max≫ to maximum data value
x resets ≪x limits≫ to data minimum-maximum
y resets ≪y limits≫ to data minimum-maximum
z resets ≪z limits≫ to data minimum-maximum
col resets ≪color limits≫ to data minimum-maximum
x linear resets ≪x scale≫ to linear
y linear resets ≪y scale≫ to linear
z linear resets ≪z scale≫ to linear
col linear resets ≪color scale≫ to linear
col black comma white resets the color values for ≪color axis≫
10 resets ≪spacing≫ to ten grid lines or tick marks per axis
x nogridtick resets ≪x gridticks≫ to no grid lines or tick marks
y nogridtick resets ≪y gridticks≫ to no grid lines or tick marks
z nogridtick resets ≪z gridticks≫ to no grid lines or tick marks
dot resets ≪type≫ to represent data as dots
white resets ≪color≫
axis resets <<<graph part colors>>> to display a white axis
label resets <<<graph part colors>>> to display a white label
gridtick resets <<<graph part colors>>> to display white grid lines or tick marks
nam label lambda resets ≪graph label≫ to λ
x label lambda resets ≪x label≫ to λ
y label lambda resets ≪y label≫ to λ
z label lambda resets ≪z label≫ to λ
col comma lambda resets <<<data color scheme>>> to λ

As well as resetting the independent strings as shown above to their initial values, the user should be able to reset the independent strings which are

forms of <graph specs> to specifications listed in <<<graph lists>>> in order to use a previously defined graph. To do this the user should be able to reconstruct a name and through the operation symbolized by rec, recall the specifications on the graph list headed by the given name. A syntax rule for this is:

<user's input string> :: = REC [<<<name>>> ; <<<graph list>>> ; ≪name≫ P [<graph specs > !]]

where the operation REC is the finding of the substring of specifications headed by the given name in <<<graph list>>> and replacing ≪name≫ rec in the user's input string, with this substring.

Consequently, the user need specify the parameters for any given display of data only once. Also, if he wishes to specify parameters for a graph that is similar to a previously specified one, he need only retrieve the former and modify it for display. By creating a new name the modified graph may then be saved on the graph list. In this fashion, the user can easily create a series of similar graphs and save them for future use.

6.7 RUBOUT PROCEDURE

It is suggested that the UAA be enabled with a special operation symbolized by rubout. Through this operation the user may be able to correct his previous actions to some degree. This would be accomplished by removing the last entry of the user's input string. A syntax rule for this operation is:

<user's input string> :: = R [<input string> <last entry> rubout; <input string>]

where <last entry> is any member of the terminal set for the UODDL.

This operation is limited. Once an independent or additive independent string has been redefined, the previous value is lost and the rubout operation cannot restore that value. Thus, the user can erase all his former actions back to the last time an independent or additive independent string was set. For example, suppose the user had created the string min-763 col. At the time he took the action col the independent string ≪min≫ was set to -763. By taking the rubout action next, col will be removed from the input string. Thus, the color axis limits will not be reset in this case. However, ≪min≫ still retains the value -763. Repeated rubout action will be ignored. Another way of understanding this is to think of the input string as being reduced to the single symbol col after ≪min≫ is set to -763. Then after the rubout action, the input string becomes an empty string.

7.0 CONCLUSIONS AND RECOMMENDATIONS

7.1 DISPLAY LANGUAGE

The Pilot Test System consists of a set of operators that are not necessarily intended to be the operators of a display language, but it is interesting to examine them in the light of the characteristics of the more conventional and formal high-level (or low-level) computer programming languages.

1. Operators are picked and then the operands are specified.

2. Combinations of operators and operands form an unlimited set of procedures (whose only end result is the generation of a data graph).

3. Input and some form of output is specified for each routine.

The primary difference between the proposed set of operators and a formal computer programming language is that there is not a method whereby a set of procedures may be named and then combined with other procedures to form more sophisticated procedures.

The creation of graphical data displays from a set of basic operators can be compared to the creation of computational sequences from a set of basic mathematical operators. For example, the end result of mathematical computation is a numeric value or set of numeric values. The end result of executing a series of data display operators is a graph of a particular data set.

The comparison is further illustrated by examining some hypothetical mathematical language which has only the operators plus (+) and minus (-) for addition and subtraction. It becomes obvious quite quickly that it would be desirable to have a multiplication operator to avoid having to specify an unreasonable number of additions. Likewise, in a graphical data display language one could have the facility for specifying a pair of vertical and horizontal lines with an appropriate intersection that would represent the axes of a two-dimensional graph. Again it becomes apparent that a single operator could be used to provide the desired set of lines and thereby reduce the amount of work required by the user.

The next possible step the designer of a language might take, given the two situations above, is to ask the question, "Is the more basic operator now required to be an element of the language?" In the case of the mathematical language, the answer is fairly obvious; in the case of the graphical

data display language, one is not quite sure that the more basic elements are required. However, two additional cases of using the basic horizontal and vertical lines easily come to mind; these include the generation of either a short or long set of lines parallel to the original axes. The first case is taken care of by including a special operator for generating a series of short lines (TIC marks), and the second case by including a special operator for generating a series of long lines (GRID). We now ask the question again, "Are the basic horizontal and vertical line operators required for generating graphical data displays?" Ideally the operator should have the ability to create subroutines like GRID and TIC given a basic set of graphical language elements. One problem encountered in this idea is the generation of mathematical relations to do things such as computing the number of tick marks and/or the spacing of grid lines.

There are several recommendations to be made for improving the way the operators function as a result of experimentation with the operators of the Pilot Test System. These recommendations are as follows:

1. The name of a graph should be automatically generated in the form xnn where nn = 00, . . ., 99 so the NAM operator would only have to be activated when the user desired to change the name.

2. A data scanning algorithm should be used to automatically set initial plotting limits that includes all points of a data set.

3. Once graphical specifications have been constructed using one data set, they should then be able to be applied to various data sets as a procedure.

4. The axes should be available as an optional feature similar to other graphical features.

5. The labeling information for the graphs should include the data set name as well as the components being plotted.

7.2 IMPLEMENTATION LANGUAGE FOR PILOT TEST SYSTEM

Based on the results of the language survey discussed in Section 4, the LISP interpretive system for the PDP-1 was used to program and execute the routines for defining a data display. The basic PDP-1 LISP system was modified to allow up to 16,384 words of storage to be utilized; also provided were LISP linkage to machine language subroutines, and the saving and loading of the interpreter on magnetic tape.

The use of LISP to write the User Action Analyzer demonstrated that a list processing language would be useful, not only for easier programming, but as a means of channeling the program development into a highly modularized structure. Also, due to the feature of dynamic storage allocation, core storage could be efficiently utilized without expending a great deal of effort in program design.

A capability of list structured languages that was not investigated during the current effort was the feature of letting the user define a set of procedures for his own special operator. The basic parts of the Pilot Test System could be used to support such an investigation. It is recommended that such an effort be pursued in the further research of formalizing a display language.

7.3 DATA STRUCTURING

The original plan for developing the Pilot Test System considered the inclusion of the data structure design postulated in Section 5. Since it was decided to take an evolutionary approach to creating data displays (i.e., define static display, manipulate static display, define dynamic or kinetic display) the first step did not require a sophisticated data structure. Also the only means of programming the algorithms for processing this data structure was the PDP-1 assembly language. In order to conduct an investigation of data manipulation and the creation of dynamic displays a programming tool such as L6 should be utilized.

BIBLIOGRAPHY

1. NASA CR-1107 , "A Report on Data Display Programming," Prepared by Wolf Research and Development Corporation under Contract NAS 5-9756-47, 1968.

2. Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System," M. I. T. Lincoln Laboratory Technical Report No. 296, January 1963.

3. Sutherland, W. R., "On-Line Graphical Specification of Computer Procedures," M. I. T. Lincoln Laboratory Technical Report No. 405, May 1966.

4. Roberts, L. G., "Graphical Communication and Control Languages," Proceedings of the Second Congress on Information System Sciences, Spartan Books, Inc., Baltimore, Maryland, 1964, pp. 211-217.

5. Raphael, B., et. al., "Language Survey," Parts I and II, A Report of the Comparison of Languages Subcommittee of the ACM Special Interest Committee on Symbolic and Algebraic Manipulation, presented at the IFIPS Working Conference on Symbol Manipulation Languages, Pisa, Italy, September 1966. Reprinted IEEE Computer Group News, September and November 1967.

6. Knowlton, K. C., "A Programmer's Description of L6," Communications of the ACM, Volume 9, Number 8, August 1966, pp. 616-625.

7. Sutherland, W. R., "The Coral Language and Data Structure," Appendix C, M. I. T. Lincoln Laboratory Technical Report No. 405, May 1966.

8. Weizenbaum, J., "Symmetric List Processor," Communications of the ACM, Volume 6, Number 9, September 1963, pp. 524-536.

9. McCarthy, J., et. al., LISP 1.5 Programmers' Manual, M. I. T. Press, 1965.

10. Evans, D. S. and Katzenelson, J., "Data Structure and Man-Machine Communications for Network Problems," Proc. of the IEEE, Vol. 55, No. 7, July 1967.

11. Van Dam, A. and Evans, D., "A Compact Data Structure for Storing, Retrieving, and Manipulating Line Drawings," Proc. of Spring Joint Computer Conference, 1967.

12. Westlake, D. A. and Chase, E. N., "Proposed Entity Table Design," Contract No. AF 19(628)-5098, Supplemental Report No. 2, April 1967.

13. Rosen, S. (Editor), Programming Systems and Languages, McGraw-Hill Book Company, New York, 1967.

14. Bernstein, M. I., "Computer Recognition of On-Line, Hand-Written Characters," Rand Corporation Memorandum, RM-3753 - ARPA, October 1964.

15. Groner, G. F., "Real-Time Recognition of Handprinted Text," Rand Corporation Memorandum, RM-5016-ARPA, October 1966.

16. Bernstein, M. I., "An Online System For Utilizing Hand Printed Input: A Progress Report," System Development Corporation Technical Memorandum, TM(L)-3052/001/00.

17. Teitelman, W., "Real-Time Recognition of Hand-Drawn Characters," AFIPS Conference Proceedings (1964 FJCC), Vol. 26, Part 1, Spartan Books, Inc., Baltimore, Maryland, 1964, pp. 559-575.

18. Davis, M. R. and Ellis, T. O., "The RAND Tablet: A Man-Machine Graphical Communication Device," AFIPS Conference Proceedings (1964 FJCC), Vol. 26, Part 1, Spartan Books, Inc., Baltimore, Maryland, 1964, pp. 325-331.

19. Teixeria, J. F. and Sallen, Roy P., "The Sylvania Data Tablet: A New Approach to Graphic Data Input," AFIPS Conference Proceedings (1968 SJCC), Vol. 32, Thompson Book Company, Washington, D. C., 1968, pp. 315-321.

20. Ledeen, K., "Harvard University Character Recognizer," private communication, February 1968.

21. Anderson, R. H., Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics, Ph.D. Thesis in Applied Mathematics, Harvard University, January 1968.

APPENDIX I

DESCRIPTION OF PILOT TEST SYSTEM

This appendix describes the graphical pilot test system which was implemented on the PDP-1. This was modeled after the system described in Section 3, and any significant deviations in design or function are noted. The operators that the experimenter will use to construct graphs of his data are described below.

A. Graphical Display Specification

The pilot system described here illustrates the set graph mode which is used to construct two-dimensional graphs. The three-dimensional graph feature was not included for this early pilot test because of its complexity to program, and the two-dimensional case is sufficient for feasibility demonstration purposes.

Once in the set graph mode, the user is presented with a set of special operators (see Figure I-1). The major changes from Section 3 are the use of 3 characters in each operator name and only 3 characters in the character set instead of the whole alphabet. This was necessary because of limited drum space (for display buffering) in which to maintain a large control overlay, and to display graphs on the rest of the scope as well. A detailed description of operators follows.

NAM — After selecting this operator the user selects any three alphanumeric characters he desires to name the graph he is creating.

DEF — This operator defines the quantities to be plotted by presenting an auxiliary control display (see Figure II-2) to the user and letting him pick his options. The user has a set of data in the form $(V_1, V_2, \ldots, V_{90})$ where each vector V_i contains 8 components $(V_{i1}, V_{i2}, \ldots, V_{i8})$. Each set of data has its own name which appears next to PLOT on this display. When the user selects PLOT, he then indicates which components he wishes to use as x and y coordinates. His choices appear in the brackets next to the data set name (JON in Figure I-2).

In addition to deciding which quantities are to be plotted, the user selects a range of values for each axis as follows. For the x range specification, the user selects X and then enters three numbers with or without minus sign. Then he

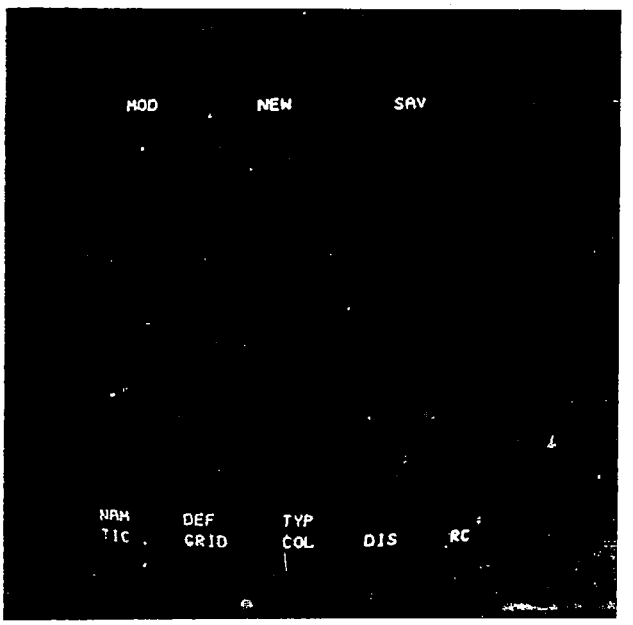


Figure I-1. Control Overlay for Set Graph Mode

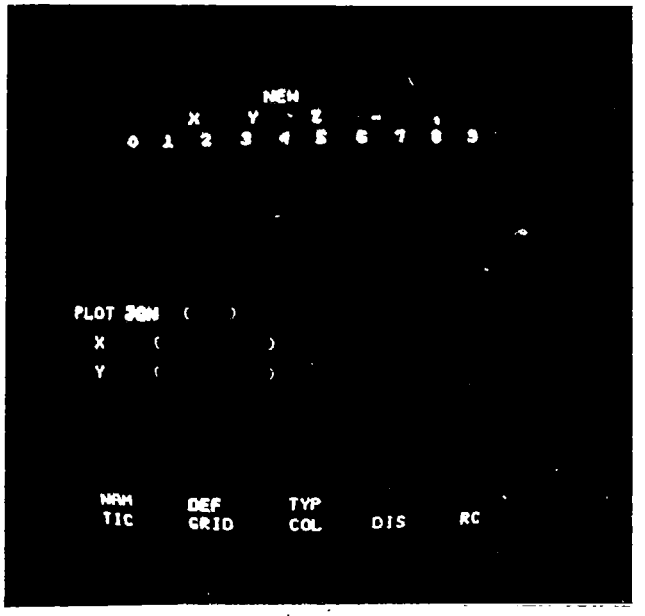


Figure I-2. DEF Mode Control Display

selects a comma and his choice for x minimum is recorded in the brackets. When he has specified x maximum and then hits another legal operator, x maximum appears next to x minimum. Range of Y is handled similarly. (See Figure I-3 for a completed set of DEF mode options.)

Note: Only integer values are used, since the floating point hardware is awkward to use in a real time environment on this configuration.

TYP — This specifies type of graph desired. Pointing at 'TYP' will give the user three options: DOT, LINE, and SEG. They represent a dot graph, a line graph, and consecutive line segments between the data points. The user selects one of the three, which turns blue to indicate his choice.

TIC and GRID — Both of these operators give the user the same auxiliary display. The only difference between these is the length of the lines (tics or grids) on the graph. If the user doesn't specify number or spacing information about one or both axes, then an automatic number of tics or grids is generated. If both these operators are not selected, then neither tics nor grids are displayed. If both are selected, then only grid lines will appear.

COL — This operator, when hit, presents to the user a set of 7 color dots, 5 control words, and the word "COLOR" (see Figure I-4). The user's color choices are restricted to 7 colors because of simplicity and discernibility. When the user has picked a color dot, the word "COLOR" will change to that color to indicate the user's current choice of color. If the user selects a control word, then that word changes color to the current color of the word "COLOR." When the user leaves the color mode, the graph elements named by the control words will be colored the same as the final colors of their control words.

DIS — Hitting this operator causes the graph to be created and displayed in the graphical area of the scope. The graph consists of data points or lines, axes and label information, and grids or tics if requested.

RC — This mode is used to recall a previous graph by hitting one of the graph names displayed on the screen. If no graphs had been saved, then this list is empty.

SAV — This operator is used to save all the user information about this graph in order that the graph may be recalled later.

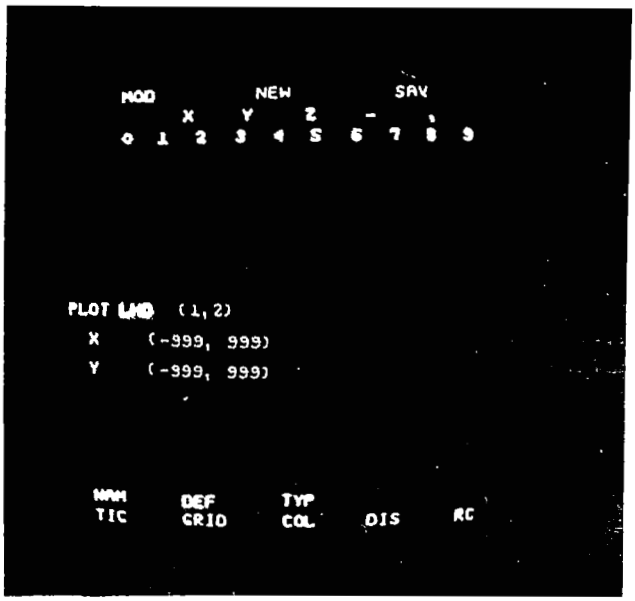


Figure I-3. DEF Mode with all Options Completed

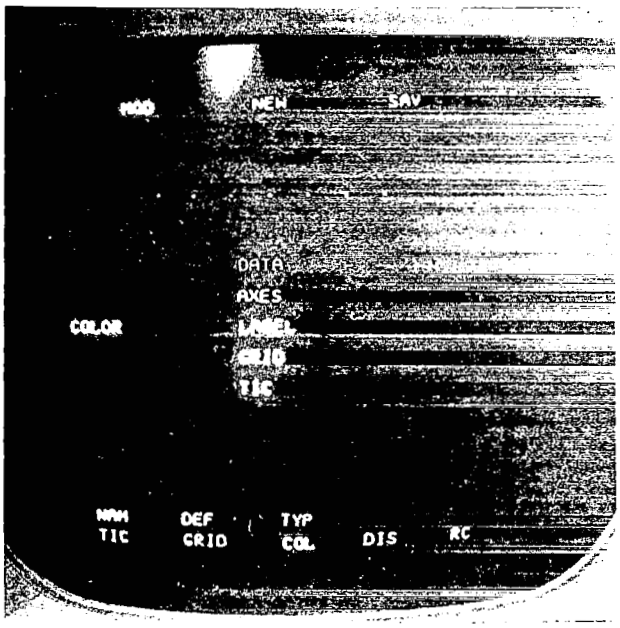


Figure I-4. Color Mode Control Display

NEW — This operator wipes out the current graph, resets the user options to "standard" and displays the available data set names. The user may select a new data set or continue to use the current one.

MOD — Selecting this operator allows the user to modify the current graph by hitting the control operators again. Nothing is reset except by user action.

B. Use of Color Coding as Man-Machine Interactive Feedback

All of the operators described above are imbedded in a system which employs color to guide and indicate meaningful user actions. Basically, four colors are used for the operators:

Red — This operator is illegal now, and picking it will result in no response.

Green — This operator is available to be selected.

Blue — This operator has just been selected and we are now working in this mode.

Yellow — This operator has been previously selected and successfully completed; although we are in another mode, this operator can be selected again.

In addition, the character set is blue when legal, and black when not legal. Graph options in the color mode, type mode, and define mode change color due to user picks. Also, some of the operators may disappear (color = black) when their presence is inappropriate or distracting. Any green operator can be activated at any time. The operator "DIS" is turned green only after both "NAM" and "DEF" have been successfully completed. If only NAM and DEF have been picked (i.e., are yellow) when DIS is hit, then the standard options for the other operators will prevail. The "standard" options include type = dot, tic = none, grid = none, and colors = white.

C. Examples of Graph Options

In Figure I-5, a data set is being displayed using the TYP option, DOT. The same data set is plotted using the TYP option, SEG, in Figure I-6. A comparison of the two figures (I-5 and I-6) shows the difference in the TYP options DOT and SEG as well as highlighting the anomalies of the data points within the indicated plotting limits. The third type of graph (shown in Figure II-7) illustrates

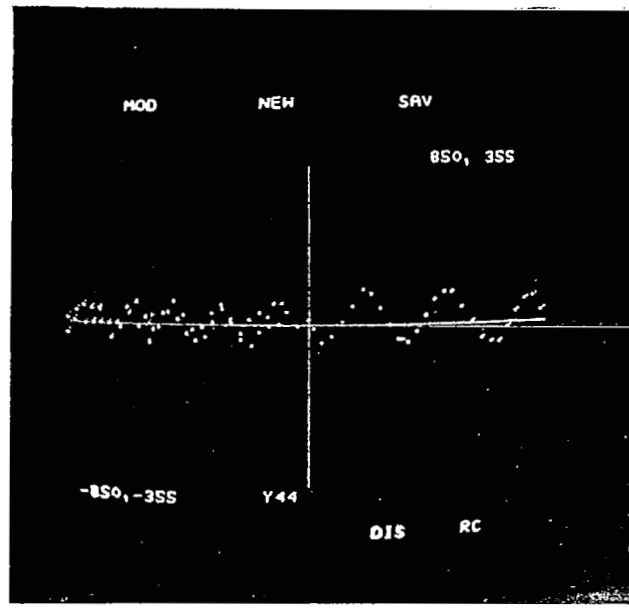


Figure I-5. Data Plot Using Dot Option

the use of the TYP option, LINE. The lines show the vertical displacement of the data points from the x-axis or the screen boundary, whichever is closer. These three figures (II-5 - II-7) also show the tick and grid lines obtained through selection of TIC or GRID mode.

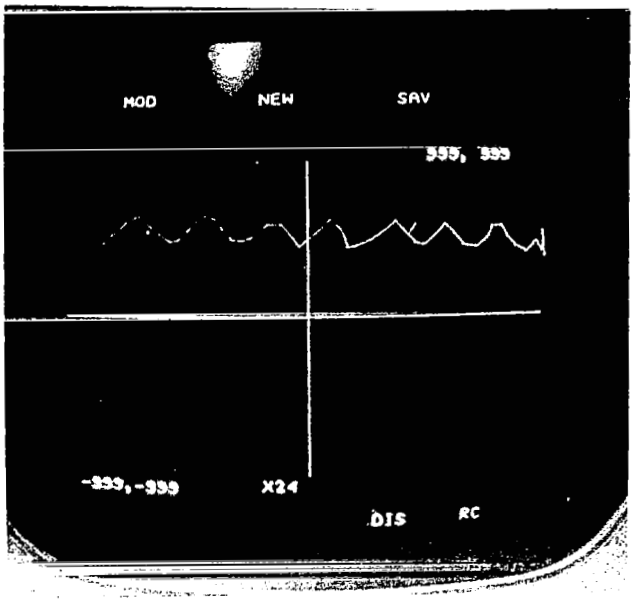


Figure I-6. Data Plot Using Line Segments

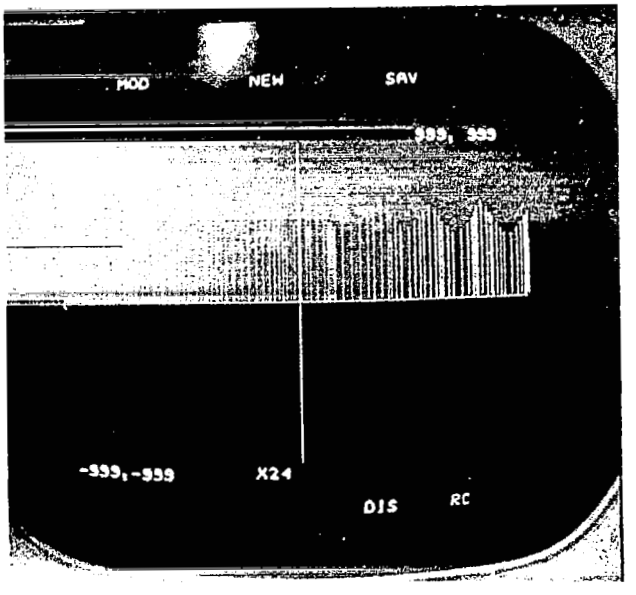


Figure I-7. Data Plot Using Line Option

APPENDIX II

USER EXPERIMENTS

A series of experiments were conducted in which persons not involved with the programming of the Pilot Test System used the system to create and modify data displays. Subjects with varying degrees of experience using CRT displays and a light pen participated in these experiments.

A. Purpose of Experiments

The reasons for conducting experiments with the Pilot Test System which only used a light pen to specify displays and control user response were the following goals:

1. Obtain user reaction and comments on working with such a system.

2. Determine the time required by various users to obtain some degree of proficiency with the system.

3. Obtain an indication of the usefulness of using color coded operators to indicate to the user his legal actions.

4. To obtain some indication of the degree of safeguards that should be provided to keep the user from making non-recoverable mistakes (fail-softness).

B. User Orientation

Most users were given a brief verbal description and demonstration of how the system operated. The remainder of the users were given the following users' guide to read and use during system operation. These latter subjects were not given any verbal assistance or demonstration of system use.

USERS GUIDE FOR GRAPH SYSTEM

Purpose - To enable the user to construct a graphical representation of any data set with little or no prior knowledge of the characteristics of the data. The system control operators are displayed in the upper and lower portions of the screen.

This is an experimental system designed to test interactive techniques for creating data displays. As such it has a great deal of flexibility and a relatively slow response time. The user is encouraged to try out any ideas that come to mind and are not explained on the instruction sheet.

NO USER ACTION WILL CAUSE SYSTEM FAILURE

NAM* — For specifying a three-letter alphanumeric name for each graph.

DEF* — For specifying the data set components to be plotted and the x - y plotting ranges. An auxiliary display is generated:

PLOT**	1 ,	8
X	-999 ,	999
Y	-999 ,	999

**Data Set Name

The underlined portions must be filled in by the user by using a numeric register which functions as follows: After picking either PLOT, X, or Y the user then uses the numeric register to fill in the desired digits for components of the data vector to be plotted and the plotting range.

NUMERIC REGISTER — When digits are picked they are displayed in what is called a numeric register which can be used to construct integer numbers from -999 to +999. Picking the - sign from the character set will make the number in the numeric register negative. Picking the - sign of the numeric register will make the number positive.

Picking the numeric register will cause it to be set to 000. Picking a non-numeric character will cause the value of the numeric register to be inserted in the variable portion of an auxiliary display that is currently being completed.

GRID — To specify an overlay GRID. Auxiliary display:

AXIS	NO.	SP.
X	•	•
Y	•	•

*These are the only operators that are required to generate a display of a data set; all others have default options specified.

Either the number (no) or spacing (sp) of the grid for each axis may be specified by picking the appropriate dot in the auxiliary display and then using the numeric register to indicate the desired spacing or number of grid lines.

TIC — To specify tic marks on the X and Y axes. An auxiliary display is generated and used as the one described above for GRID. The default option is for no tic marks. If the tic operator is picked and no spacing or number of marks is indicated, the system will automatically generate tic marks on each axis.

TYP — To specify the type of data plot desired. An auxiliary display is generated providing three choices:

DOT
LINE
SEG

The user picks his choice. Default option is for point plotting (DOT).

COL — To specify colors for individual portions of the graph. An auxiliary display is generated:

Color Dots
Red . . . . . White
DATA
AXES
LABEL
GRID
COLOR TIC

A color is indicated by picking one of the color dots and then the item that is to be colored. The word COLOR indicates active color dot.

RC — To recall a previously created graph. The names of previously created graphs are displayed and the one to be recalled is picked. It may then be modified or displayed.

DIS — Whenever the current graph being recalled, created, or modified is to be displayed, this operator is picked.

MOD — To modify the currently displayed graph.

SAV	To have the currently displayed graph saved for later reference.
NEW	Picking this operator will generate a display of the data set names.* After identifying the data set of interest, a new graph may be created.
	Color coding is used to guide the user's choices of operators as follows:
RED	Illegal operator: picking such an operator will have no effect.
GREEN	Legal operator: when picked will change color and perform the desired operation.
BLUE	Legal operator: indicates the user's current choice of operator. (The alphanumeric characters are always blue.)
YELLOW	Legal operator: after an operator has been picked, and the required information supplied, it is turned yellow during the creation of a graph. A yellow operator may be picked again to change a previous choice.

The above explanations were intentionally made as brief as possible with the idea of being used as a user's instruction sheet and not as a complete system description.

C. Background of Subjects

The subjects were chosen from the following experience categories with at least one subject from each.

1. Graphical display programming including color CRTs with a degree in science or engineering.

2. A non-programmer who is currently using color display equipment for data analysis and has a degree in science or engineering.

3. A non-programmer who did not have experience in the use of CRT display equipment but has a scientific or engineering degree.

*Data sets consist of 8 component vectors and any two components may be used to generate an x - y graph.

4. A programmer/analyst with a business degree.

5. A non-programmer that has neither display experience nor a scientific or engineering background.

D. Observations of Subjects

In general a subject's use of and reaction to the system could be directly related with his educational background and working experience. The subjects who had experience in conducting data analysis in the past by either manual or computer methods seemed to grasp the use of the display specification operators quite easily and would then create a meaningful representation of a given set of data.

The subjects given an oral description and demonstration of system use were able to perform with about the same or better capability as the user that started by reading the users guide and experimenting on his own for a period of 20-30 minutes. The confidence of subjects in either case improved markedly after they had specified and viewed their first data display.

Most of the users had not used a light pen extensively in the past but they seemed to adjust easily to its idiosyncrasies. The most laborious part of light pen use was the specification of alphanumeric information, as might be expected. Although the Pilot Test System was fairly unsophisticated, its free form features of allowing the user to build his own unique display seemed to appeal quite universally and almost every experiment exceeded the amount of time allotted to the subject.

E. Subjects' Comments

The comments made by the various users should be useful for future considerations in the design of an interactive data analysis system.

A consolidated summary of the comments by the users is presented below:

1. The feedback of what operator was picked with the light pen takes longer for some operators than for others.

2. There should be a positive indication of when a data display is finished being generated.

3. The current values for specifying the number of grid lines in a graph should be saved and displayed similar to the way the DEF mode operates.

4. A keyboard is a more useful device for the hierarchical specification and manipulation of display entities. This user felt that a sufficient number of options could not be provided for a complex system by the sole use of light pen.

5. The alphanumeric symbols for specifying display names and parameters should be located at the bottom of the CRT screen.

6. The maximum plotting ranges specified in the DEF mode should be automatically set for an initial quick-look at a data set.

7. Explanations on the user's guide were not sufficient for some of the operators.

8. The entering of alphanumeric information by picking characters with the light pen was felt to be a very tedious task.

F. Conclusions

In general, the users found the Pilot Test System to be useful, interesting, and easy to use in spite of certain limitations such as the slow reaction time of the LISP interpretor and the typically awkward light pen. The fact that a new user of the system gained familiarity and proficiency in using the system in a relatively short time span (20-30 minutes) indicates that using a common working surface for input and display is a desirable feature for an interactive system.

The usefulness of color coding operators based on their status or legality was helpful once a user obtained a feel for how the system operated. It was first thought that this feature would be most useful for instructing the new user of his possible legal actions. The experiments proved that this assumption was wrong.

The primary safeguard against illegal user actions was code the illegal operators in red and then have the system not respond if one of these operators was picked. There were, however, some instances when the user had legal operators that could cause the loss of previously defined procedures if they were picked at the wrong time. In order to avoid such a situation the user should have a default option on at least one level of his actions.