Display techniques for interactive text manipulation

by CHARLES H. IRBY

Stanford Research Institute
Menlo Park, California

INTRODUCTION

The Augmentation Research Center (ARC) at the Stanford Research Institute (SRI), has been developing for several years a computer-based on-line system called NLS. NLS is part of ARC's research on enhancing the intellectual effectiveness of people.[3,5,6,7,10,12,13,15] Central to the developments to date is highly interactive text manipulation using chiefly display terminals.[16,17,18] The NLS system supports a range of display terminals (from expensive text/graphics displays to inexpensive Alpha Numeric displays[1,2]) and typewriter terminals. The NLS program runs as a subsystem within a TENEX time-sharing system on a DEC PDP-10 computer.[9]

NLS is a program of about one hundred thousand instructions and about eight programmers are involved in its continued development and maintenance. Since the program has been and will be under development for several years, considerable attention is given to the employment of good software engineering practices.

NLS provides a general purpose interface to any of a large number of specialized capabilities that the user may draw upon during his work. Certain capabilities, such as text manipulation and communication with others, are important to almost any type of intellectual work, and, thus, they have received a large amount of our development resources. NLS provides the user with a consistent and coherent command language interface while allowing him to access diverse capabilities. The system is used intensely in the day-to-day work of about fifty people, some of whom access the system through the ARPA NETWORK.[11,14] These people are writers, managers, engineers, analysts, and programmers.

In addition to very flexible text editing and viewing, NLS provides the user with facilities for communication, publication-quality formatting control, numerical calculation, specialized user-supplied editing and viewing, and programming support (such as a built in debugging system and direct access to several compilers).

For a more complete description of NLS and its applications, the reader should consult References 3 and 5.

Figure 1 describes the basic structure of the NLS application program. This paper is primarily concerned with the capabilities that the Display Terminal Interface provides to the rest of the application program.

Based on the command language grammar and the user's input, the command language interpreter invokes various manipulators to modify data structures and, if appropriate, formatters to map these data structures into specified rectangular portions (called "windows") of the display screen for the user to see. User input in Figure 1 represents character input, coordinate input, and selection input (based on coordinate input).

A manipulator is that set of routines that manipulates data structures of a certain type, say type "A". An example might be the data structures used to represent a hierarchical structure that is applied to the textual information contained in the user's files. Some of the data structures are contained in the user's files; others are used to maintain user or system state information and characteristics. Such a manipulator might be applied to any of several instances of type A data structures or might always be applied to a specific instance.

A formatter consists of those routines that map a data structure of a certain type into a rectangular "window", say "a", on the display screen. Such a formatter might invoke subformatters to handle subparts of the data structure, and it might be applied to a particular instance or to any of several instances of such a data structure. A formatter might be applied to a specific window or applied to any of several windows.

In order to minimize the number of changes that will have to be made to the screen, a formatter may compare what is currently shown in the window to what is desired. To facilitate this, a formatter maintains a data structure to reflect the current contents of the window. Alternatively, the formatter may simply clear the window and format the new data into it. The size of the window (the number of characters wide and lines high) is available to a formatter from the Display Terminal Interface.

The Display Terminal Interface is that set of routines that provides the application program with primitive operations for the manipulation of and interaction with a conceptual display terminal. This interface allows the application program to support physical displays with quite different characteristics. The protocol between the terminal and

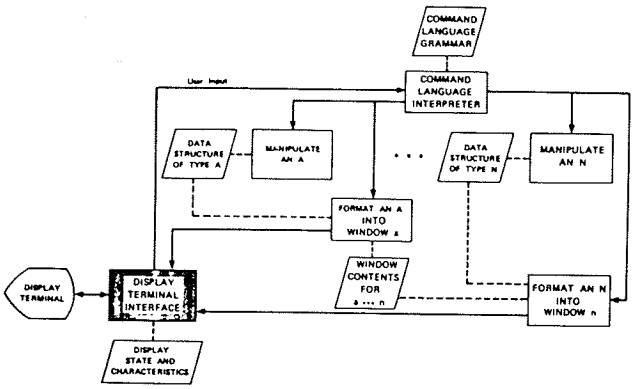

Figure 1—Basic structure of NLS

the Display Terminal Interface may vary with the terminal type.

Figure 2 illustrates the window organization of a typical NLS display screen. Figure 3 shows an actual screen organized in this way. Figures 4 through 8 show other organizations on various physical display terminals.

In developing the graphics portion of the system, we wished to make use of the fairly well-known notions of "structured" display images and "virtual" display terminals in order

(1) To support a wider range of terminals without major changes to the application program.

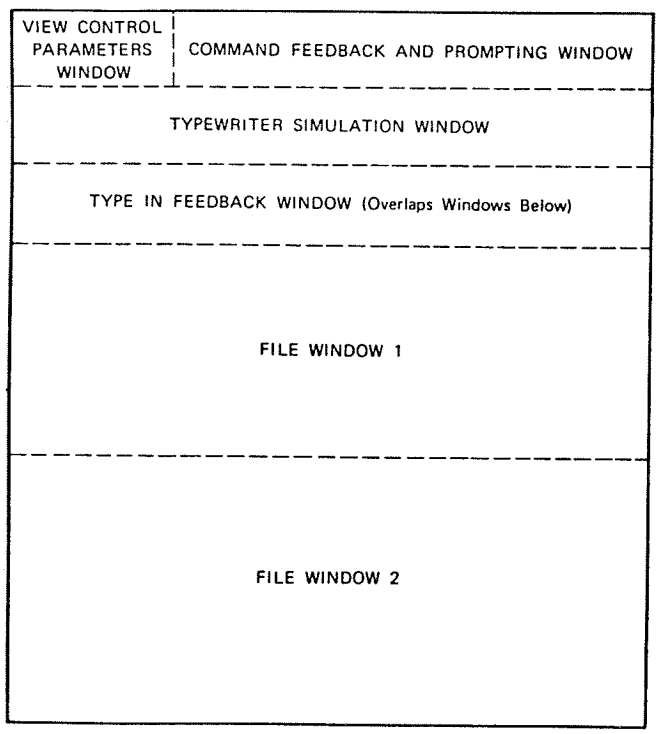

Figure 2—A typical NLS display screen subdivided into windows—See Figure 3

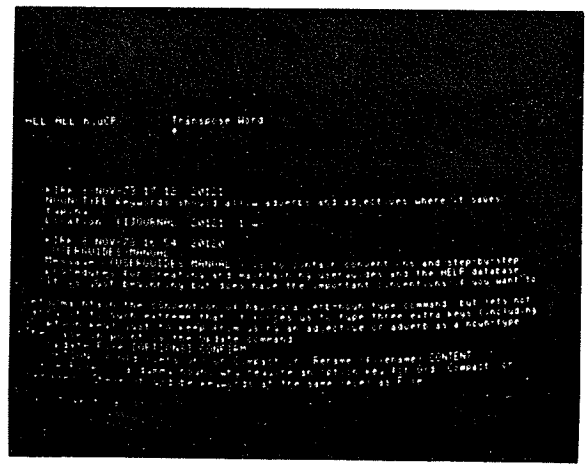

Figure 3—Photograph of NLS display which corresponds to Figure 2. Typewriter simulation window and type in feedback window are empty. The display terminal is a Delta Data 5200 with a Line Processor.[1,2] Note highlighted text in upper file window, operands selected by the user for the Transpose Word command. Text may be moved from one file to another by selecting operands in separate file windows

(2) To minimize the amount of information to which a particular formatter must have access in order to modify a certain portion (window) of the display.

By "structured" display images, we mean display images subdivided into a structure (usually hierarchical or sequen-

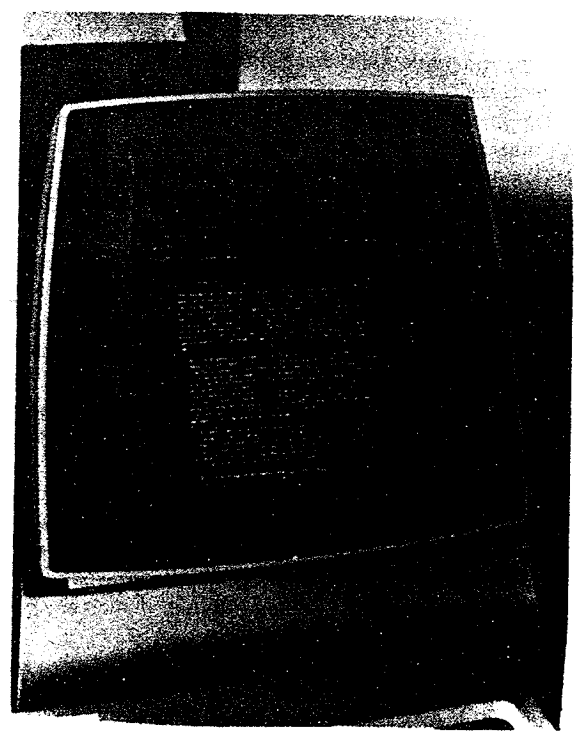

Figure 4—Photograph of IMLAC PDS-1 NLS display terminal with one file window

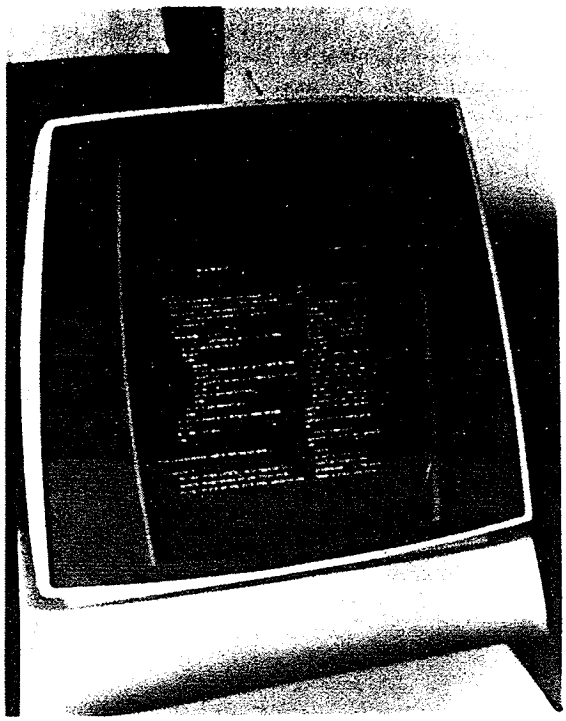

Figure 5—Photograph of IMLAC PDS-1 NLS display terminal with two columnar file windows with two different files being displayed

tial), such that the parts of the structure can be modified (such as deleted, moved, or replaced) independently from the rest of the display image. By "virtual" display terminal we mean a display terminal manipulated by an application program so that conceptual display properties can be mapped by interface routines into appropriate commands for the physical display being supported.

However, in attempting to apply these techniques to text display and manipulation, we discovered that there are

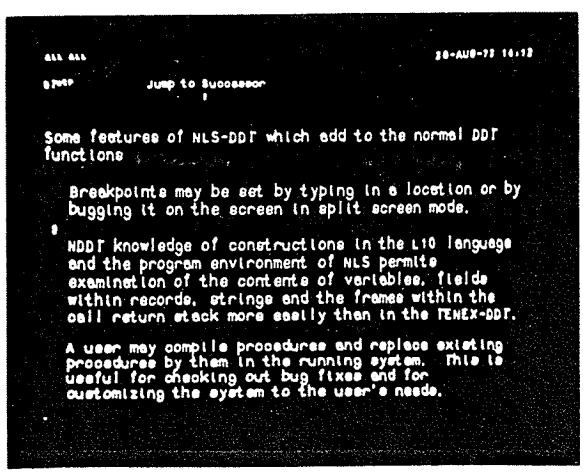

Figure 6—Photograph of local display terminal screen with one file window showing documentation for debugging program

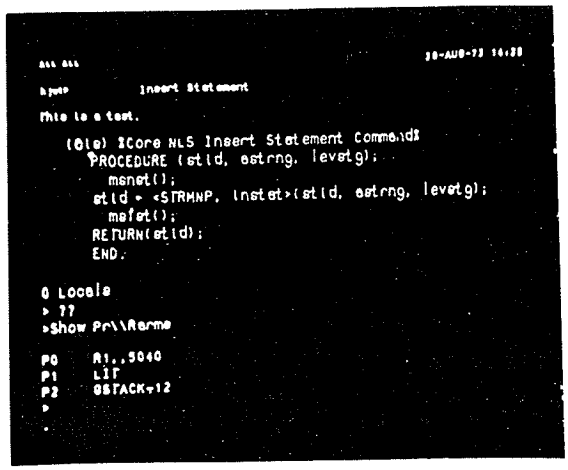

Figure 7—Photograph of display screen with one file window and one typewriter window (lower portion of screen). User may interact with NLS on upper portion of screen or with debugger on lower portion

some underlying differences between pictorial graphics, on which these techniques work quite well, and textual graphics. These differences forced us to develop a slightly different conceptual model for text displays. This paper reports what we now know about the differences and the conceptual model we have developed.

SOME FUNDAMENTAL DIFFERENCES BETWEEN TEXTUAL AND PICTORIAL GRAPHICS

Although pictorial and textual graphics are similar in most respects, there are some problems unique to textual graphics

(1) On most displays, only characters of certain sizes and spacing are acceptable to the human user (or can be displayed at all).

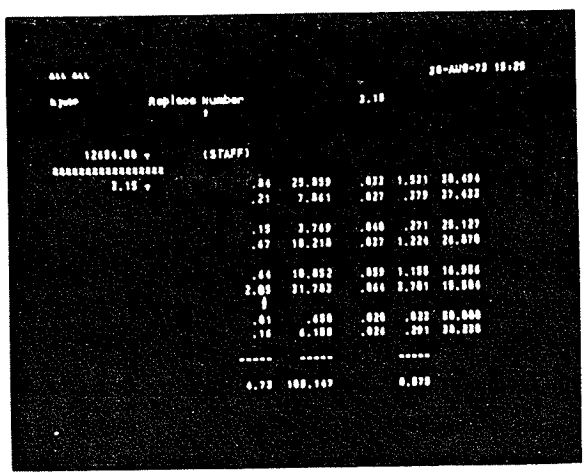

Figure 8—Photograph of local display screen showing use of numerical calculation features with a transaction history window on left and user's file on right

(2) Often, characters can only be displayed at certain coordinate positions with a predetermined spacing between characters. Thus, mapping a virtual coordinate system onto a physical screen may be difficult. Most displays with fixed-spaced character fonts can be thought of in terms of a character-grid coordinate system, which is not necessarily the same as its pictorial coordinate system.
(3) In general, text cannot be scaled, rotated, or translated by arbitrary amounts (as can most pictorial images).
(4) In order to control text formatting, the application program must know the character-grid coordinate system(s) of the physical display.

In order to do its job effectively (from the user's standpoint), the application program must be able to determine the usable character sizes and fonts and their associated character-grid coordinate systems for the physical display it is supporting.

These fundamental differences forced us to develop the conceptual model discussed in the following section.

A CONCEPTUAL MODEL FOR TEXT DISPLAYS

Our requirements for a conceptual model of a text display are as follows:

(1) Characteristics of the physical display should be isolated or parametric. A range of physical displays must be supported with minimal impact to the application program. We have found the "isolation of knowledge" to be an essential software engineering principle to effect long term reliability, flexibility, and maintainability of a large software system.
(2) Separate parts of the application program must be able to manipulate independently the text on portions of the screen.
(3) The user must be able to "select" text on the screen by means of some type of "pointing" device. By a pointing device, we mean a device that is capable of transmitting coordinate data to the application program (e.g., mouse,[17] stylus and tablet, joy stick) in response to some user action, such as depressing a button or a key. The pointing device should be coupled to the display in such a way that it gives the user some indication (e.g., the visual indication provided by tracking the device with a cross-hair) of what he is likely to select if he makes a selection. When the user wishes to select some text on the screen, he moves the pointing device so that it (or its displayed tracking spot) is near the desired text and takes the appropriate action to cause the coordinates to be transmitted. The application program should then determine which text is nearest the coordinates that were input and show the user what it found (e.g., by highlighting the selected text as in Figure 4). This must be done in such a way that the user can "back up" (say, by depressing some other button) and retry the selection.
(4) The ability of two or more users to "share screens" must be provided. We find great value in the ability of two or more people who are geographically distant to run NLS using display terminals through the ARPA Network and share screens. By this we mean both see the same image on their screens and both can control the application program that manipulates the image. The situation is analogous to several people standing together at a blackboard, where all can see what each writes. This sharing is greatly facilitated, of course, by a telephone connection. By this means, distributed people can work together on such things as reports, designs, papers, proposals, and computer programs. Video projectors also allow distributed meetings.
(5) If the application program needs to use the same portions of the screen for different purposes, it is very convenient for the application program to be able to suppress the display of part of the image and later to be able to restore it to sight. This is useful since most display screens are quite small, in terms of the number of readable characters they will support, and portions must often be used for several purposes. For example, the same portion of the screen might be used for the display of information from the user's files and for the display of status messages or the feedback of user input. The suppression capability allows the application program to overlap windows and use the physical screen space to best advantage without having to dedicate portions of the screen to infrequently used purposes.

Figure 7 shows a situation where the display of a file text window has been suppressed in order for the user to interact with a debugging program in that portion of the screen. Figure 7 also shows feedback of user input (the text "this is a test") in a sequential window that extends to the bottom of the screen. As the user types more text, lines will be suppressed in the file text windows as need to avoid superpositioning.

(6) The application program must be able to draw the user's attention to some text on the screen (e.g., make it blink or increase its intensity—see Figure 3).
(7) Because of the typewriter-like interaction modes of most modern time-sharing systems, typewriter simulation should be possible on a portion of the screen when running the application program in display interaction mode (for system broadcast, error, or warning messages from the time-sharing system). We have found this to be very valuable to the user. This portion of the screen must, in general, be dedicated to this purpose because of the asynchronous nature of these messages. In Figure 7, the debugging program is interacting with the user through typewriter-

simulation on the bottom portion of the screen. The text on the upper portion of the screen is unaffected by the scrolling (simulating the behavior of Carriage Return and Line Feed) which takes place during typewriter simulation on the lower portion.

In order to meet these requirements, we have developed a conceptual model of a display terminal. The reader is referred to Figure 1, to the appendix of this paper, and to other referenced material (especially References 2 and 19) for additional details. The primary characteristics of the conceptual model are as follows.

Windows and Strings

The display screen is divisible into rectangular, possibly overlapping "windows". Windows may be invisible or visible, random or sequential. Sequential windows behave like typewriter simulations (text is scrolled though them). Random windows contain character strings which can be manipulated (moved, replaced, deleted) independently. Individual strings may or may not be selectable. Text in selectable character strings may be selected by the user via his pointing device as operands to application program commands. The terminal initially has only one sequential window that covers the whole screen and is called the "default typewriter" window.

The application program is expected to allocate windows for various types of information display to the user. Some of these windows are for the purpose of command specification feedback to the user and others are for the display of information contained in the user's files.

Basic terminal modes

The terminal can be in one of two basic modes: (1) "typewriter" mode and (2) "display" mode. In "typewriter" mode, all display windows except the default typewriter window are invisible, the default typewriter window is visible, and coordinate input is disabled; the terminal acts like an alpha-numeric display simulating a typewriter terminal. In display mode, the default typewriter window is invisible and coordinate input is enabled; the application program controls which windows are visible.

Pointing device interaction

It is assumed that in addition to character input, the terminal also transmits coordinate information along with at least certain characters. In formatting character strings that are selectable by the user (usually representing text from the user's files), the formatters construct a data structure associating each character string with the data element that it represents. When the user subsequently selects a character on the screen, the coordinates that were input are mapped by the display terminal interface, using mapping data that it maintains, into a window-identifier, string-identifier, and character count. This character and/or neighboring characters may then be "highlighted" on the screen for the user's benefit. The window-identifier, string-identifier and character count are converted by the application program, using the data structure just discussed, into data element identifiers appropriate for its use.

Sequential windows

We assume a situation where the user has only one terminal that must behave like a typewriter terminal at times and like a true two-dimensional display terminal at other times. Thus, sequential typewriter windows are very important. Any text that is received by the display that is not in the context of a display command is "scrolled" through the current typewriter window. The effect of characters like Carriage-Return and Line-Feed are simulated. We expect that, when an application program is initialized, it allocates a small sequential window somewhere on the screen and makes it the typewriter window. Thus, any error messages, system broadcast messages, terminal "linking",[9] and so forth, can be seen by the user while using the terminal in display mode.

Device specific parameters

When the Display Terminal Interface (see Figure 1) is initialized by the application program, it determines (via monitor calls or interaction with an "intelligent" terminal) enough about the display characteristics to manipulate the physical display. It returns to the application program the character-grid coordinate systems for the available character sizes of the terminal. The rest of the application program is then parameterized, on the basis of these values.

To make all of this work, we must make certain assumptions about the display (and any associated processing capability it might logically possess).

It is mandatory that:

(1) we can treat the screen like a large character grid and write characters at arbitrary positions on the grid (providing that we do not write past the edge of the screen),
(2) there is some way of mapping our conceptual display primitives, described in the Appendix of this paper, into the primitive operations of the physical display,
(3) there is some way of writing text in a mode such that it stands out from the rest (e.g. blink, reverse video, underline),
(4) there is some way of highlighting existing text on the screen in such a way that when the highlighting is removed, the original text will look just as it did before it was highlighted (This may be the same as (3) above), and
(5) there is a coordinate input device such that the cur-

rent coordinates will be input with at least certain characters and such that it can be tracked on the screen.

It is desirable but not mandatory that:

(1) there is some way of accomplishing the typewriter window capability (although this is not a must, the capability is certainly useful to the user),
(2) various (fixed spaced) fonts and character sizes are available for the terminal (we plan to extend the model to include proportionally spaced fonts in the future), and
(3) the (intelligent) display terminal is capable of responding to an interrogation command from the Display Terminal Interface. This capability is optional, since the user can supply the information instead. However, this latter approach is not very desirable or reliable.

Our operating system makes assumptions about the type of terminal that one is using. If these assumptions are incorrect (for example, if one has a display rather than a typewriter terminal), then the user must communicate this to the operating system via a command. If the terminal is intelligent and can respond to an interrogation, the user simply specifies this, and when the Display Terminal Interface is initialized, it sends the terminal an interrogation command, to which the terminal responds with its characteristics. Otherwise, the user must supply any needed information about the display terminal or its characteristics must be assumed by the application program.

THE MOUSE AND KEYSET AS IMPORTANT AIDS TO DISPLAY INTERACTION IN TEXT EDITING

Although they are very simple devices, we have found that the mouse and keyset, combined with a standard typewriter-like keyboard, form a very balanced and useful set of input devices for two-dimensional text manipulation.[4,8,16,17,18] The mouse is used for pointing and for special function input; the keyset and keyboard are used for character input. The mouse is a small device that has two perpendicularly mounted potentiometers, to which are attached wheels that roll and slide in proportion to the direction of movement, and three buttons. It is an easy "pointing" device to use and causes the user little or no fatigue. It can be used on almost any flat surface, usually a desk top.

The keyset consists of five long keys, similar in shape to white piano keys. The user depresses several keys in unison to input a character. When the thirty-one possible combinations are combined with shift buttons on the mouse, the user is able to completely duplicate the standard keyboard, while keeping one hand on the mouse, ready to point to operands for commands typed in from the keyset. When more than a few characters are to be input, the user removes his hands from the mouse and keyset and uses the typewriter-like keyboard.

The three buttons on the mouse, if depressed and released without intervening characters from the keyset, have additional functions, the more interesting of the seven being:

To select some text on the screen or give final confirmation to begin the execution of a command,

To back-up command specification, to allow the user to redo whatever he just did (e.g. select something else on the screen or retype his last character),

To abort the current command specification and return the user to the beginning of command specification, and

To allow the user to modify the parameters which control how his information is presented to him. He may do this in the middle of specifying a command.[3]

For a more extensive discussion of these devices, the reader's attention is directed to References 2 and 4.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to other members of the ARC staff for their help in the development and implementation of the conceptual model discussed in this paper. In particular, he would like to thank Ken Victor for his help, criticism, much implementation work, and for maintaining the official documentation for the IMLAC protocol.[19] In addition, the author would like to thank Don (Smokey) Wallace for implementation help, Don Andrews for his work on the Line-Processor,[2] and L. Peter Deutsch, of XEROX Palo Alto Research Center, for helping to develop the protocol for supporting IMLAC PDS-1 display terminals and for writing an IMLAC program to implement it.

The work reported here was and is currently being supported primarily by The Advanced Research Projects Agency (ARPA) of the Department of Defense, and also by the Rome Air Development Center of the Air Force and the Office of Naval Research.

REFERENCES

1. Hardy, M. E., "Micro Processor Technology in the Design of Terminal Systems," under Preparation for the *Proceedings of the IEEE COMPCON*, 1974, SRI-ARC Catalog Item 20185.
2. Andrews, D. I., "Line Processor: A Device for Amplification of Display Terminal Capabilities for Text Manipulation," prepared for the *Proceedings of the National Computer Conference*, May 1974, SRI-ARC Catalog Item 20184.
3. Engelbart, D. C., R. W. Watson and J. C. Norton, "The Augmented Knowledge Workshop," *AFIPS Proceedings National Computer Conference*, June 1973, 38 p., SRI-ARC Catalog Item 14724.
4. Engelbart, D. C., "Design Considerations for Knowledge Workshop Terminals," *AFIPS Proceedings National Computer Conference*, June 1973, 38 p., SRI-ARC Catalog Item 14851.
5. Engelbart, D. C., *SRI-ARC Summary for IPT Contractor Meeting*, San Diego, 8-10 January 1973, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, 7 January 1973. 8 p., SRI-ARC Catalog Item 13537.
6. Engelbart, D. C., *Coordinated Information Services for a Discipline- or Mission-Oriented Community*, Stanford Research Institute, Aug-

mentation Research Center, Menlo Park, California, paper given at Second Annual Computer Communications Conference, San Jose, California, 24 January 1973, 12 December 1972, preprint, 13 p., SRI-ARC Catalog Item 12445.
7. Augmentation Research Center Staff, *Online Team Environment: Network Information Center and Computer Augmented Team Interaction*, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, RADC-TR-72-232, 8 June 1972, 266 p., SRI-ARC Catalog Item 13041.
8. Savoie, Robert, *Summary of Results of Five-Finger Keyset Training Experiment*, Project 8457-21, Stanford Research Institute, Bioengineering Group, Menlo Park, California, 29 March 1972, 4 p., SRI-ARC Catalog Item 11101.
9. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, A Paged Time Sharing System for the PDP-10," presented at *ACM Symposium on Operating Systems Principles*, 18-20 October 1971, Bolt Beranek and Newman Inc., 15 August 1971, SRI-ARC Catalog Item 7736.
10. Engelbart, D. C., *Network Information Center and Computer-Augmented Team Interaction*, Interim Technical Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, RADC-TR-71-175, AD 737 131, July 1971, 104 p., SRI-ARC Catalog Item 8277.
11. Roberts, L. G. and B. D. Wessler, *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques, Washington, D. C., May 1971, SRI-ARC Catalog Item 7750.
12. Engelbart, D. C., *Experimental Development of a Small Computer-Augmented Information System*, Annual Report, 15 April 1970, 15 April 1971, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, 15 April 1971, 8 p., SRI-ARC Catalog Item 8616.
13. Engelbart, D. C. and Staff of ARC, *Advanced Intellect-Augmentation Techniques*, Final Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, CR-1827, July 1970, 212 p., SRI-ARC Catalog Item 5140.
14. Engelbart, D. C., "Intellectual Implications of Multi-Access Computer Networks," paper presented at *Interdisciplinary Conference on Multi-Access Computer Networks*, Austin, Texas, April 1970, Preprint, 12 p., SRI-ARC Catalog Item 5255.
15. Engelbart, D. C. and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, Vol. 33, 1968, 15 p., SRI-ARC Catalog Item 3954.
16. Engelbart, D. C., W. K. English and J. F. Rulifson, *Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research*, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, AD 843 577, April 1968, 180 p., SRI-ARC Catalog Item 9697.
17. English, W. K., D. C. Engelbart and M. A. Berman, "Display-Selection Techniques for Text Manipulation," in *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, March 1967, p. 5-15, SRI-ARC Catalog Item 9694.
18. English, W. K., D. C. Engelbart and Bonnie Huddart, *Computer Aided Display Control*, Final Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, CR-66111, N66-30204, July 1965, 104 p., SRI-ARC Catalog Item 9692.
19. Staff of ARC, *IMLAC User's Guide*, Stanford Research Institute, Augmentation Research Center, unpublished, available upon request.

APPENDIX—PRIMITIVES OF THE CONCEPTUAL MODEL OF A TEXT DISPLAY

The primitive operations that the Display Terminal Interface provides to the application program are listed here.

For window Manipulation:

WINDOW-ID ← ALLOCATE-WINDOW (X1, Y1, X2, Y2, CHARACTER-SIZE, FONT, TYPE)

Function: Allocates a rectangular window of the specified type (random or sequential) and position. Establishes default character size and font for the window.

Arguments:

X1, Y1: screen coordinates of upper left corner of window.

X2, Y2: screen coordinates of lower right corner of window.

CHARACTER-SIZE: default character size for this window.

FONT: default font for this window.

TYPE: sequential or random

Returns:

WINDOW-ID: unique identifier for this window (to be used in subsequent commands).

DEALLOCATE-WINDOW (WINDOW-ID)

Function: Deallocates the specified window.

CLEAR-WINDOW (WINDOW-ID)

Function: Deletes contents of window and removes image from the screen.

INVISIBLE-WINDOW (WINDOW-ID)

Function: Makes the contents of the window invisible (no image on the screen).

VISIBLE-WINDOW (WINDOW-ID)

Function: Makes the contents of the window visible (image appears on the screen).

TYPEWRITER-WINDOW (WINDOW-ID)

Function: Makes the specified (sequential) window the typewriter window. All "unescorted" characters (not within a display command) will be scrolled through this window. These characters can also be scrolled through the default typewriter window so that the user can see them when the terminal is returned to typewriter mode.

For Character String Manipulation:

STRING-ID ← WRITE-STRING (WINDOW-ID, X, Y, CHARACTER-SIZE, FONT, HIGHLIGHT, SELECTABLE, CHARACTERS)

Function: write the specified string in the window, with the specified properties at the specified position.

Arguments:

WINDOW-ID: unique identifier for a window.

X, Y: window coordinates of the first character of the string.

CHARACTER-SIZE: Use specified character size for this string or use window default.

FONT: Use specified font for this string or use window default.

HIGHLIGHT: If specified, highlight this string (make it stand out to user).

SELECTABLE: If specified, characters in this string may be selected by the user via the SELECT-CHARACTER primitive.

CHARACTERS: the characters to be displayed.

Returns:
STRING-ID: unique identifier for the string within this window.

REPLACE-STRING (WINDOW-ID, STRING-ID, X, Y, CHARACTER-SIZE, FONT HIGHLIGHT, SELECTABLE, CHARACTERS)
Function: Replaces the specified string in the specified window by the characters specified. If the X, Y coordinates are not specified, the current position is used. FONT and CHARACTER-SIZE may be defaulted to the old values for the string or to the window defaults. If HIGHLIGHT is specified, the string is made to stand out from normal text on the screen.

MOVE-STRING (WINDOW-ID, STRING-ID, X, Y, CHARACTER-SIZE, FONT, HIGHLIGHT, SELECTABLE)
Function: Move the specified string to the specified position within the window.

DELETE-STRING (WINDOW-ID, STRING-ID)
Function: Delete the specified string from the specified window.

INVISIBLE-STRING (WINDOW-ID, STRING-ID)
Function: Make the specified string invisible to the user (no image on the screen).

VISIBLE-STRING (WINDOW-ID, STRING-ID)
Function: Make the specified string visible to the user (image on the screen).

CLEAR-STRING (WINDOW-ID, STRIND-ID)
Function: Same as REPLACE-STRING with null string.

For Sequential Window Manipulation:

APPEND-TEXT (WINDOW-ID, CHARACTERS)
Function: Append the specified characters to the specified sequential window. Carriage Return and Line Feed characters are simulated within primitive and are automatically inserted to avoid characters exceeding the right edge of the window.

For Highlighting Characters:

MARK-CHARACTERS (WINDOW-ID, X1, X2, Y)
Function: Highlight the characters from position X1, Y to X2,Y in the specified window, such that the mark can be removed with REMOVE-MARK and the original characters will be unchanged. It is desirable, but not mandatory, for the user to be able to read characters that are marked by this primitive.

REMOVE-MARK ()
Function: Remove the last mark put on the screen with MARK-CHARACTERS.

CLEAR-MARKS ()
Function: Remove all marks put on the screen with MARK-CHARACTERS.

For Cursor Manipulation:

SET-CURSOR (CHARACTERS)
Function: If possible for this display terminal, set the primary cursor (the one that tracks the user's pointing device) to the specified characters.

PLOT-SECONDARY-CURSOR (X,Y, CHARACTERS)
Function: Plot a secondary cursor at screen position X,Y using the characters specified if possible. This must be done in such a way that the original text on the screen is not destroyed. This primitive is used in screen sharing.

For User Input:

CHARACTER ← READ-CHARACTER ()
Function: Read the next character input from the terminal.

(X,Y) ← READ-CURSOR-COORDINATES ()
Function: Read the next (screen) coordinates iuput from the terminal.

SEND-COORDS-WITH-CHARACTERS ()
Function: Begin sending cursor (screen) coordinates with (at least certain control) characters.

DONT-SEND-COORDS-WITH-CHARACTERS ()
Function: Stop sending cursor coordinates with any characters.

TIME-INTERVAL-COORD-INPUT (TIME-INTERVAL)
Function: Begin reporting cursor coordinates periodically (when they have changed), independent of user actions, for use in screen sharing.

For User Selection of Text on the Screen

(WINDOW-ID, STRING-ID, CHARACTER-COUNT, X′, Y′) ← SELECT-CHARACTER (X,Y)
Function: Given the screen coordinates X,Y, find the nearest selectable character on the screen.
Returns:
WINDOW-ID: The unique identifier for the window containing the string that contained the selected character.
STRING-ID: The unique identifier for the string containing the selected character.
CHARACTER-COUNT: The index into the string identified by STRING-ID of the character that was selected.
X′, Y′: The window coordinates of the selected character.

WINDOW-ID ← SELECT-WINDOW (X,Y)
Function: Given the coordinates X,Y, return an identifier for the nearest window containing selectable character strings. Such windows should not overlap.

For Batch Processing Display Commands:

PROCESS-COMMANDS (DISPLAY-COMMANDS-LIST, WINDOW-ID)
Function: Given a list of display commands (like those described above), perform the operations all at once on the display in a manner appropriate to the actual display.

For Error Messages:

OUTPUT-ERROR-STRING (CHARACTERS)
Function: Output the error message in a manner appropriate to the display.

For Determining Display Characteristics:

PARAMETERS ← INTERROGATE-DISPLAY ()
Function: Determine the usable character sizes, fonts, and character-grid coordinate systems for the display.

The "normal" character size and font are also indicated. Execution of this primitive also initializes the Display Terminal Interface routines to work with the actual display.

For Basic Mode Switching:

TYPEWRITER-MODE ()

Function: Put the terminal in typewriter mode. Make all windows invisible except for the default typewriter window and disable coordinate input.

DISPLAY-MODE ()

Function: Restore terminal to display mode. Make default typewriter window invisible, make any windows that were visible prior to the last TYPEWRITER-MODE command visible again, and enable coordinate input.

For Resetting the Terminal to Its Initial State:

RESET ()

Function: Reset the display terminal to its initial state, simulating a typewriter-like terminal with no windows allocated and not sending coordinates with any characters.