UTILITY SUBROUTINES

This section describes five different tasks and the I/O utility subroutines which were developed to meet their requirements. The basic problem associated with all of these tasks is that of requesting various types of information from the graphic console operator. Hence, the emphasis is on the implementation of the various input devices associated with the graphic console. The graphic console display CRT (cathode ray tube) is used to indicate what type of information is required.

Alphanumeric Input

Subroutines names, variable names, titles for recorded film output, etc. constitute one type of information frequently required from the graphic console operator. This information is most conveniently handled by the programmer in the form of BCD character strings. Thus, the task here was to provide a facility for requesting and accepting these strings. Two input devices are appropriate: the alphanumeric keyboard and the card reader (see Figure 1).

In addition to the primary requirement that the I/O subroutine for this task be simple for the programmer to use, two other requirements were felt to be important. First, there was the need to give the graphic console operator immediate feedback which would permit him to verify that each character has been entered correctly. Concomitant with this was the need to provide the operator with a means of correcting erroneously entered characters.

Figure 1. The 7960 Alphanumeric Keyboard and Card Reader.
The subroutine RINSE (Request for Information Subroutine) was developed for this task. To use this subroutine, the programmer specifies three items of information: (1) the symbolic name (i.e., location in core) of an array which contains the request message in BCD format, (2) the maximum number of BCD characters which the operator is allowed to enter, and (3) the symbolic name of an array in which to store these characters. Figure 2 shows a typical set of source language statements for utilizing the RINSE subroutine. The DIMENSION declaration is used to reserve space for an array at compile time. The VECTOR VALUES declaration is used to preset an array (i.e., vector) at compile time. The EXECUTIVE statement generates a call to a subroutine.

In Figure 3 the request message in the example above is shown as it would appear on the display CRT upon the execution of RINSE.

Assume that the graphic console operator wishes to enter the subroutine name: ABC. At this point, the operator can either type in the subroutine name at the alphanumeric keyboard or insert a card in the card reader. If he elects to type in the name, each character will be added, one at a time, to the display. If he inserts a card in the card reader, characters will be added to the display sequentially as they are read from the card. The console operator can intermix these two modes of input. Figure 4 shows the display as it would appear after the subroutine name has been entered.

The operator can delete the last character in the string by depressing the BACKSPACE key or delete the entire string by depressing the RESTART key (see Figure 1). He may then enter more characters. In this manner, erroneous characters may be corrected. Once the input is satisfactory, the operator depresses the END key, the display is terminated and the BCD character string is passed back to the calling program.

**Numeric Input**

Another type of information which the programmer may require from the graphic console operator is numeric data. The task here is one of providing a facility for initializing and/or modifying data variables. Two subroutines were developed to meet this requirement. The difference between these two routines stems from the two basically different ways in which data variables can be defined: as elements of an array or as a set of distinct variables which may be widely scattered throughout memory. In the first case, each variable is referenced by giving the name of the data array and the subscript of the particular item. In the second case, each variable has its own unique name.
In addition to the three task requirements mentioned in connection with the RINSE subroutine (i.e., ease of use, visual feedback to permit verification, and an error correction procedure) there was the additional requirement that the operator be able to enter numeric data in either floating point, integer or octal mode. For this task, the only input device which has been implemented is the alphanumeric keyboard.

The subroutine SETDA (SET up Data Array) permits the inspection and modification of items in a data array. To use SETDA, the programmer specifies four items of information (see Figure 5): (1) the BCD name of the data array, (2) the length of the data array, (3) the symbolic name (i.e., location in core) of a second array which defines the mode (integer, floating point or octal) of each item in the data array, and (4) the symbolic name of the data array. If each item in the data array has the same mode, the mode need only be specified once for the whole array.

Figure 6 shows the display which will be generated upon execution of SETDA, as indicated in Figure 5.

At this point, the operator can modify the value of the current item, step the display to the next item in the data array, define the subscript of any item in the data array which he wishes to see displayed next, or terminate the subroutine.

The operator enters a new value for the current data item by using the alphanumeric keyboard. The visual feedback and error correction procedures are identical to those described for the RINSE subroutine. Figure 7 shows how the display would appear during this process. The operator can cause the new value of 27 to be stored in DATA(0) by depressing the END key.

```
VECTOR VALUES NAME = DATA(0)
VECTOR VALUES DATA1 = 2.37477K
VECTOR VALUES MODE = 314LFS.KK
EXECUTE SETDA(NAME,314,MODE,DATA1)
```

Figure 5. Source Language Statements for SETDA Subroutine.

If the operator tries to store a number of the wrong mode, the display changes to that shown in Figure 8.

If the operator wishes to step the display (Figure 6) to the next item in the data array, he depresses the alphanumeric key labeled "→" and the upper portion of the display changes appropriately. If, on the other hand, the operator wishes to "move" the display directly to any item in the data array, he first depresses the "=" key. The display then changes to that shown in Figure 9 and the operator enters the subscript of the desired item.

If the operator tries to enter a subscript which is too large, the display changes to that shown in Figure 10.

Control is returned to the calling program when the operator depresses the END key from the display indicated by Figure 6.

The second subroutine (SETPAR) developed to facilitate numeric input permits the inspection and modification of a set of distinct data variables. The programmer uses this subroutine in essentially the same manner as SETDA. However, the format in which the current status of the data variables is displayed was altered in an attempt to gain insight and experience in the presentation of information. The information display technique used by the SETDA subroutine can be characterized as the "player-piano" approach: the data items appear as if they were listed sequentially on a long strip of paper which is being moved back and forth past a one-item viewer under the opera-
tor’s control. SETPAR uses a “page” approach as the information display technique. This technique treats the data variables as if they were listed, up to 16 to a page, in a loose-leaf notebook. As each “page” is presented to the graphic console operator, he has the options of modifying the values of any data items listed on that page, turning to the next “page”, or “closing the book” (i.e., terminating the subroutine).

Figure 11 shows a typical “page” display.

If the operator wishes to change the value of any data variable in the displayed list, he uses the alphanumeric keys labeled “↑” and “↓” to move the pointer to the desired data item and then enters the new value of this item using the alphanumeric keys exactly as in the SETDA subroutine. To “turn the page” the operator depresses the “,” key and to terminate the subroutine he hits the END key.

**Positional Data Input**

Since the GM Research Laboratories DAC-I system was developed specifically to investigate the field of graphic data processing, it was necessary to provide a capability for presenting the graphic console operator with a display of graphic information and receiving positional input from him relative to this display. The position indicating pencil provided the basic hardware capability necessary to meet this need. The requirements for this task were: (1) to display the graphic information specified by the calling program, (2) to determine the position of the pencil relative to this display, and (3) to return this position to the calling program when the pencil was removed from the screen.
Because of the very general nature of graphic information, no attempt was made to assist the programmer in formating his display. To increase the flexibility of the I/O utility subroutine designed for this task (PEN2) an option was added which allows the graphic console operator to terminate the display by depressing either an alphanumeric key or a program control button. In any case, however, the calling program is fully informed as to the condition which terminated the display.

Figure 12 shows the source language statement necessary to execute PEN2. The 2nd and 3rd arguments define the location and size of the array containing the graphical information. The 4th argument describes the condition which terminated the display. The 1st argument provides information pertaining to the pencil.

Upon execution of PEN2, a display of the specified information begins. At this point, the graphic console operator can depress an alphanumeric key, a program control button, or bring the position indicating pencil into contact with the display CRT. If a key or button is depressed, the display is immediately terminated and control passes back to the calling program. If the pencil is touched to the screen, a small cross appears superimposed on the display. This cross defines the position of the pencil relative to the display and provides the operator with visual feedback. The visual feedback is particularly important here because of severe parallax problems. If the pencil is moved across the screen, the cross will follow. When the pencil leaves the screen, the display is terminated and the last location of the cross is returned to the calling program. Figure 13 shows the pencil being used to point to a region on a grid.

This subroutine has proven to be very flexible and easy to use, resulting in a wide variety of applications. One such application of PEN2 was in the development of a higher level I/O utility subroutine permitting the entry of decision type information.

Decision Information Input

The task associated with the entry of decision information is to present the graphic console operator with a set of possible actions available to the controlling program and to allow him to select an ordered subset of these actions.

The I/O subroutine CHOICE was developed for this task and requires three items of information from the calling program: (1) the number of alternatives to be presented, (2) the symbolic name (i.e., location in core) of an
array containing the BCD messages describing the alternatives, and (3) the symbolic name of an array provided for the list of selected alternatives (see Figure 14).

The execution of this subroutine begins with a display of the alternatives, as shown in Figure 15.

The operator may now select his first alternative by pointing to the appropriate field with the voltage pencil and then removing the pencil from the screen. Immediate visual verification of the choice is provided by the digit 1 at the right hand side of the selected field as shown in Figure 16.

The operator proceeds in this manner to complete his selection of the desired alternatives. When he is done, he selects the field labeled “DONE” and control is returned to the calling program. If the operator makes a mistake or changes his mind, he can “erase” his last selection by selecting the field labeled BACKSPACE.

SOURCE LANGUAGE STATEMENTS

In the first section of this paper, five subroutines were described which are representative of a class of subroutines whose primary objective is to facilitate the on-line input of various types of information. In the following section, we describe three source language statements which provide the programmer with the capability of generating various types of on-line output in an easy, efficient, and flexible manner. In the development of these statements, an effort was made to conform (insofar as possible) to the precedents established by the normal I/O source statements appearing in the NOMAD language. It was felt that, in so doing, the resulting statements would be much easier for the programmer to use.

Display Format Statement

“DISPLAY FORMAT” was designed to permit the programmer to use the graphic console display CRT in much the same way as the on-line printer was used in “the good old days.” The implementation of this “on-line print” capability provides the programmer with a very convenient way of presenting a wide variety of information to the graphic console operator.

The maximum amount of information which can be displayed by DISPLAY FORMAT (i.e., its unit record) is 480 characters. These 480 available character positions are in the form of 20 lines on the CRT with 24 character positions per line.
The DISPLAY FORMAT statement has a list and a format associated with it. The list gives the values and items to be displayed and the format gives the display pattern to be used. The format is composed of a sequence of format specifications of the form $C_w$. The character $C$ is the control character and defines the type of operation or conversion to be performed. In general, $w$ refers to the number of character positions associated with the operation defined by $C$. If, in any format specification, the field width $w$ is zero, that format specification and its associated list item (if any) are ignored.

There are three types of format specifications available for the conversion of numeric data. Octal and decimal integers are generated by using the $K_w$ and $I_w$ specifications respectively. The $F_{w,10}$ specification provides for floating point variables which will be displayed as either floating point decimals (e.g., .2E-3) or fixed point decimals (e.g., .0002) depending on which form produces the fewest characters for display.

There are two types of format specifications available for the conversion of BCD information. The $A_{w,10}$ specification is identical with that used in other NOMAD I/O statements. In particular, one word (six characters) will be picked up from the list and right justified or truncated on the right, depending on whether $w \geq 6$ or $w < 6$. The $H_w$ specification differs from ordinary I/O statement usage in that the Hollerith information is specified in the list rather than in the format.

The specification $X_w$ provides $w$ blanks in the display. The specification $/w$ results in a downspace of $w$ lines. An automatic downspace of one line results when the number of characters on a line reaches 24. The specification $P_w$ ($w$ has no particular significance in this case unless it is 0) resets the current character position to the first character position of the first line. This specification has proven very useful when putting out columns of information which comes from several different arrays in memory.

The specification $R_w$ causes the succeeding portion of the format to be repeated, if necessary, until the list is exhausted ($w$, again, has no particular significance unless it is 0). If the specification $R_w$ does not appear anywhere in the format, the entire format will be repeated until the list is exhausted.

The form of the “DISPLAY FORMAT” statement and a sample format is shown in Figure 17.

Perhaps the most radical departure of the DISPLAY FORMAT statement from standard NOMAD I/O statements involves the use of its second “argument” (i.e., the item called INOUT in Figure 17).

INOUT will, in general, be the first cell of a short array in which the program using DISPLAY FORMAT can:

- specify the status light configuration during and upon termination of the display,
- specify the input devices which the graphic console operator will be permitted to use in terminating the display,
- receive a description of the condition which caused the termination of the display.

The graphic console display shown in Figure 18 corresponds to the DISPLAY FORMAT statement of Figure 17.

Because of its flexibility and ease of use, the DISPLAY FORMAT statement has been a valuable aid in the development of a cooperative man-machine problem-solving system.

Record Format Statement

The RECORD FORMAT statement provides a means of producing hard copy alphanumeric output through the use of the recording feature of the image processor. The form of the RECORD FORMAT statement is identical to that of DISPLAY FORMAT (i.e., RECORD FORMAT FMT, INOUT, L1, L2, L3, ...). RECORD FORMAT recognizes all of the format specifications.
The unit record for RECORD FORMAT is 12,000 characters per film frame. These 12,000 available character positions are in the form of 100 lines of 120 character positions each.

With RECORD FORMAT, INOUT is a single cell and is used by the calling program to specify the desired film train operations associated with the recording. Appropriate bytes in INOUT define which film train is to be used, how many frames are to be advanced before and after the recording is performed, and when developing of the film frame is to occur (if at all).

RECORD FORMAT enables the programmer to use the image processor as he would use an off-line printer with the added advantage that his turnaround time is measured in minutes rather than hours. The output can be developed immediately and moved to the project station of the image processor for on-the-spot review and then placed on an aperture card for hard copy reproduction. The hard copy output is very similar to that produced by an IBM 1403 printer.

**Generate Format Statement**

The DISPLAY FORMAT and RECORD FORMAT statements enable the programmer to easily produce alphanumeric output. However, it is frequently necessary to produce either displays or recordings which contain a combination of graphic data and alphanumeric characters of different sizes (see Figure 19). In addition, it is desirable to have the capability of altering a small part of a display without having to regenerate the whole display. The value of such a capability is best illustrated in the operation of the CHOICE subroutine (see Figures 15 and 16). The GENERATE FORMAT statement was specifically designed to meet these needs.

With respect to formats, input lists, and statement form (i.e., GENERATE FORMAT FMT, INOUT, L1, L2, L3, …), GENERATE FORMAT is identical to the previous two statements. However, instead of producing output directly (as is the case with DISPLAY FORMAT and RECORD FORMAT), GENERATE FORMAT provides the programmer with the array of CRT coordinates needed to produce a display of the alphanumeric information. In addition, GENERATE FORMAT enables the programmer to specify character size and line spacing (i.e., the user can define his own unit record).

By executing GENERATE FORMAT several times with different unit record specifications and combining the resultant arrays of CRT coordinates with an array of coordinates representing graphic data, the programmer can, through the use of a basic display or recording subroutine, produce output of the type shown in Figure 19.

The special features of GENERATE FORMAT were utilized in the I/O utility subroutines SETDA, SETPAR, and CHOICE to quickly and efficiently produce sequences of displays in which only portions of the basic display are altered. For instance, Figure 16 can
be produced from Figure 15 simply by adding the three coordinate pairs necessary to generate the character 1 to the large coordinate array for the basic display.

The DISPLAY FORMAT and RECORD FORMAT statements provide the programmer with an easy and efficient method of utilizing the output capabilities of the 7960 system. The GENERATE FORMAT statement rounds out the output software package by providing the additional flexibility needed to handle the situations described above.

SUMMARY

Any attempt to implement a system which stresses cooperative man-machine interaction must concern itself primarily with the problem of man-machine communication. Furthermore, the man-machine communication will typically take place in a wide variety of situations and at many levels of problem discourse. Attempts to define all possible communication situations and develop specific programs for each will lead to a tremendous amount of redundancy and duplication of effort. It is possible to avoid this problem by realizing that all man-machine communication must pass through the I/O hardware interface. Thus, by providing an extensive, flexible, and powerful I/O software capability, the presence of the hardware interface need not concern the programmer and he can devote his whole attention to the more significant aspects of man-machine communication. The software described in the body of this paper has proven to be an immensely powerful tool which has enabled programmers to rapidly design and evaluate the wide variety of communication techniques necessary to any man-machine problem-solving system.

ACKNOWLEDGEMENTS

The material described in this paper is not solely the work of the authors. Significant contributions were made by other members of the programming staff. In particular, we would like to acknowledge the contributions of C. Richard Lewis of the Computer Technology Department, GM Research Laboratories, and Martin Weinrich.

REFERENCES


* Now at the University of Michigan.
A LINE SCANNING SYSTEM CONTROLLED FROM AN ON-LINE CONSOLE

Fred N. Krull and James E. Foote
Research Laboratories, General Motors Corporation, Warren, Michigan

INTRODUCTION

Direct graphical input is one of the newest and most exciting sources of digital computer input. Programming techniques and hardware are beginning to appear which are designed to automatically process graphic information. This paper describes an experimental system which has been designed to facilitate the digitizing of line images. The equipment which is used for this purpose is an IBM 7960 Special Image Processing System, consisting of graphic console, image processor, and a modified data channel. The image processor (Figure 1) contains a programmable Cathode Ray Tube (CRT) scanner, a CRT recorder, a 35mm camera, and film processing equipment.

The graphic console (Figure 2) contains devices to communicate with the operator and to control the image processor.

Both of these devices are tied directly to a computer via a modified data channel.

The principal objective of this portion of DAC-I (Design Augmented by Computers) project was to utilize the full capabilities of the image processor, graphic console, and digital computer to digitize a variety of line images to a high degree of accuracy. Program system objectives dictated that line widths .05% of image size (.01 inches on 20x20 inch document) must be detectable to an over-all system accuracy of ±.05% of image size. Thus, this system was designed to provide a rapid, versatile, and accurate method for converting graphical data to digital form. While it provides only for the digitizing of lines, a natural outgrowth of the work will be a library of pattern and character processors to be used by programmers in much the same manner as card and tape input/output (I/O) routines are used now.

The application requirements of the DAC-I project dictated that one of two approaches be used for the analysis and processing of a two dimensional graphical form:

a) Algorithms are specified which prescribe the rules for analyzing a graphical image. This approach assumes a structure for each characteristic image, and relies upon the availability of individual pattern processors to detect image components.

b) No fixed structure is assumed for the input document. Minimal restrictions are placed on image quality only. Emphasis is placed upon providing elementary functions which an operator may call upon and combine in order to process a complex form. All decision capability and selection of functions is left to the operator who is controlling the scanning device from an on-line console.

The second approach was selected since at this point in the project it was not possible to specify the structure of the large variety of input documents anticipated. We felt, for instance, that the system would be used to digitize