A FORTRAN V INTERACTIVE GRAPHICAL SYSTEM

Alan C. Reed, et al

Utah University

Prepared for:

Advanced Research Projects Agency

3 April 1968

Technical Report 4-4

Alan C. Reed
D. E. Dallin
Scott T. Bennion

A FORTRAN V INTERACTIVE GRAPHICAL SYSTEM

April 3, 1968

COMPUTER SCIENCE
Information Processing Systems
University of Utah
Salt Lake City, Utah

Advanced Research Projects Agency · Department of Defense · ARPA order 829
Program code number 6D30

The system described provides the capability of transforming an ordinary FORTRAN V program into a highly interactive program. The converted program makes the user, the display scope, and teletype console all an integral part of the execution of the program. At present, the system allows the user to modify or retrieve current values of variables in the program or to effect transfer of execution to various statement numbers when commands are typed. A group of display oriented programs is also described which greatly simplify the job of displaying the solution and aids for finding the solution of the problem. A SWAP method is used to provide sharing of the central processor between interactive programs and the batch programs.

Before proceeding with a discussion of the details of the graphical system, it would seem only proper to first ask and then attempt to answer the following two fundamental questions:

What are the advantages of interaction?

Why interact with FORTRAN?

In regard to the first question, the solution of any problem on a computer, especially one involving complex numerical computation, requires an extreme degree of closeness between the programmer and the algorithm. Interactive graphics places the programmer and users of the program in a most advantageous position. The execution of the program may be monitored and action taken to help the program find the solution, or to find it more quickly. In addition, since the user can guide the program, a much shorter logical path to the solution will be taken than could feasibly be coded into a batch program. The increase in speed of problem solution may quite conceivably be in the form of minutes versus days, or days versus months.

The answer to the second question is also highly pragmatic. FORTRAN is a popular language which has been in wide use for several years. Most numerically oriented problems have been and are presently being solved on computers with FORTRAN programs. The desirability of interacting with this type of program has previously been pointed out. It is clear then, that FORTRAN, at present, would be the most desirable choice.

At the University of Utah, an interactive graphics laboratory has been established and is continually being expanded to provide

the computer scientist the necessary tools for research and development into man/machine systems. This laboratory is separate and distinct from the computer center and the Univac 1108 computer and peripheral equipment.

The equipment in the graphics laboratory now includes:

An on-line Univac 1004 card reader/printer and 1108 console monitor which permit the user to read-in, execute and obtain printed results of his programs while in the graphics lab.

A PDP-8 computer connected on-line to the Univac 1108. The PDP-8 acts as a controller for the graphics equipment and serves as the communications link between this equipment and the 1108.

A model 35 teletype which serves as the interactive keyboard.

An IDI display scope which may be used to provide a window into the solution space of the problem and may be combined with the teletype to form an interactive console with keyboard and display.

Other equipment and equipment features such as the Sylvania tablet, IDI light pen, pen matching and tracking, light buttons, etc., are not described in this report. We thought it best to limit our discussion to our current work and the equipment and other associated software necessary for the use of our present system.

The interactive routines described herein are taken from and

built upon those described in the Graphics System Technical Report (1). Those routines described in Technical Report 4-1 are located on FASTRAND file $$GS$$ and the others on file $UUCC$. Both FASTRAND files must be read into the program file through the CUR operations (2) prior to execution.

In addition to generating displays and handling teletype input and output, the set of routines includes those which establish the modes of operation and interaction. The first such routine is RELOAD. This routine must be called only once and before any others described below. Its function is to initialize the graphics system and to type out a sign-on message on the teletype console.

Hardware interrupts from the teletype, light pen, and Sylvania tablet are saved by the graphics system in the 1108. They are then recognized and activated as software interrupts by the user's program through subroutine IDLE. The name IDLE is a misnomer, for its function is to check if any interrupts have been received and then to transfer control according to the type of interrupt. If it is a character interrupt from the teletype, then program control will be passed to the instruction set specified by a call to the subroutine CHRINT (N,$STMT).

The characters, STMT, denote a FORTRAN statement number defined in the user's program at which the interrupt will be processed when N characters have been typed. For other interrupt conditions, we refer you to the Graphics System Technical Report (1).

Before transferring control to the interrupt location, the subroutine link parameter generated by CALL IDLE is pushed into the top of a stack. Upon completion of handling an interrupt, the subroutine INTRET must be called to unstack the top subroutine link parameter. This parameter is then used to return program control to the statement following the corresponding CALL IDLE statement. Refer to page 12 for an example of the use of CHRINT and IDLE.

Note that stacking the subroutine link parameter allows the user to declare new interrupt statement numbers if necessary and to call IDLE within his interrupt handler. Care must be taken in unstacking the link parameters in the correct sequence through calls to INTRET.

Swapping provides for use of the 1108 only when needed and allows it to be released to the normal batch processing when not needed. The 1108 EXEC II system has been modified to time share the interactive graphics user and batch processing stream for economic reasons. Whenever a call is made to subroutine SWAPER(N), the user's program is read out of memory onto a reserved region of an FH-432 drum and the current or next batch stream program is read off of drum into memory and is executed. At any time, the graphics user may swap out the program currently being executed and swap his own program back in. If N = 0, the user's program is swapped into memory and executed when the swap character is typed at the teletype console. The swap character is declared by a call to the subroutine SWPCHR('C) where C denotes the swap character (usually a carriage return character, 0-7-8 punch). If N = 1, the swap occurs whenever any interrupt occurs and a swap character need not be declared.

After the initial swap of the interactive program onto drum, the EXEC II system resumes the normal batch processing mode and assumes, for the most part, that the swapped out job is terminated. This results in restricted servicing of the interactive program when subsequently swapped back in. Normal input and output is not allowed and will cause an ENDSWP condition on the 1108 teletype if done other than through the subroutine ARPAIO (1). FORTRAN print/plot output may be written on a reserved FASTRAND file and later retrieved. This mechanism is initialized by a subroutine SWAPIO, which must be made prior to the first call to the SWAPER routine. All print/plot specifications subsequent to the call to SWAPIO will cause the corresponding fieldata print lines and plot commands to be written out on the FASTRAND which can later be retrieved by executing the program FSTDMP. Care must be taken to dump this file before the next graphics user also writes on it. Since, under EXEC II, only one graphics user may be active at a time, this restriction should not cause too much difficulty.

Teletype Routines

The following package of subroutines constitutes an extension to FORTRAN V in which two new types of variables may be defined. The first type is called an "Interaction variable", the value of which may be changed or simply retrieved by the user. The second type is called a "Command variable" which, when typed, causes transfer of execution to a statement number in the program. Command variables are also interaction variables.

A command variable is an interaction variable which is further declared in a call to subroutine JUMPS.

```
NAMELIST/NAME/VAR1, VAR2, ..., VARN
   .
   .
   .
CALL SETLST
READ (5,NAME)
   .
   .
   .
CALL JUMPS ('NAME',$STMT1, $STMT2, ..., $STMTM)
```

In the above example, $M \leq N$. The call to JUMPS declares the first M variables appearing in namelist NAME to be command variables. When VAR1 is typed, $1 \leq I \leq M$, the program transfers to STMT1 in the program.

Finally, whenever the teletype is to be interrogated, a call to subroutine TTY is made. Usually, the call will be the character interrupt processing location specified in a call to CHRINT as demonstrated by the following example:

```
LOGICAL FLAG
DIMENSION I (10,10)
NAMELIST/NAME/PLOT, STOP, A, GO, I, J, FLAG    @ DEFINE
CALL SETLST                                    @      INTERACTION
READ (5,NAME)                                  @                VARIABLES
CALL JUMPS ('NAME', $10, $20, $30, $40)        @ DEFINE COMMAND VARIABLES
CALL CHRINT (1, $5)                            @ INTERRUPT TO 5 WHEN 1
                                                 CHAR IS TYPED.
2 CALL IDLE                                    @ WAIT FOR
```

Namelist is a FORTRAN V feature which allows unformatted input and output of the variables declared in the list (3). In order to accomplish this, certain information regarding each variable must be kept in core at execution time. It is the presence of this information which makes the interaction with variables possible.

Unfortunately, a namelist name may only appear in a READ or WRITE statement. For this reason, a call to subroutine SETLST must be followed by a READ or WRITE statement in order to make the namelist table of information available to the system. The following example will illustrate:

```
NAMELIST/NAME/VAR1, VAR2, ..., VARN
.
.
.
CALL SETLST
READ (5,NAME)
```

SETLST anticipates that a READ statement of the form shown will immediately follow the subroutine call. The information given in the READ statement may be thought of as the argument list for SETLST. The subroutine returns to the next statement following READ.

The function of SETLST is to declare VAR1, ..., VARN as available for interaction and takes into account their individual variable types and dimensions. SETLST may be called more than once with various namelist names.

An <u>interaction variable</u> is one which has been declared in a NAMELIST where the name of that namelist has been provided to subroutine SETLST at execution time.

```
      GO TO 2                          @          INTERRUPT
    5 CALL TTY                         @ INTERRUPT OCCURRED, INTERROGATE TTY
      CALL INTRET                      @ RETURN TO IDLE LOOP
   10 CALL PLOTER (I, J)               @ 'PLOT' COMMAND WAS TYPED.
      CALL INTRET                      @ RETURN TO IDLE LOOP
   20 CALL EXIT                        @ 'STOP' COMMAND WAS TYPED.
   30 NEWVAL = A**3 + .5               @ NEW VALUE OF 'A' WAS TYPED
      CALL INTRET                      @ RETURN TO IDLE LOOP
   40 CALL COMPUT (NEWVAL, FLAG)       @ 'GO' WAS TYPED.
      CALL INTRET                      @ RETURN TO IDLE LOOP
```

The above program would call IDLE repeatedly until a character interrupt caused a transfer out of IDLE to statement number 5 where TTY would then be called. TTY would wait until a carriage return (↵) was typed at which time it would scan the input line. What happens next would depend upon the line that was typed. Some examples will illustrate:

Input	EFFECT
PLOT ↵	Transfer to statement 10
STOP ↵	Transfer to statement 20
A=3.5↵	Value 3.5 is stored in A then transfer to statement 30.
A=3 ↵	Value 3.0 is stored in A then transfer to statement 30.
A ↵	Transfer to statement 30.
I(1,1) = 37 ↵	Value 37 is stored in I then return to statement after CALL TTY.

INPUT	EFFECT
J = 2 ↵	Value 2 is stored in J then return to statement after CALL TTY.
I(J,J) = 40 ↵	Value 40 is stored in I(J,J) where the current value of J is used, then return to statement after CALL TTY.
I(1,25) = 2 ↵	Characters '25' underlined with '*' (subscript out of range) then wait for more input.
A = ↵	Type out current value of A. Do not transfer. Return to statement after CALL TTY.

The above list includes most of the common forms of input but is by no means exhaustive. The following rules govern the use of TTY:

Values must agree in type with variable names as defined in the program. The only exception is that a floating point variable may be given the value of 3 in lieu of 3. or 3.0.

Floating point input may be either of the F or E format type.

Complex values are input as two floating point values separated by a comma.

Subscripts may be either integer constants or non-subscripted interactive integer variables.

If a label and an equal sign are typed but not a value, the program assumes retrieval is wanted and types out the current value of the variable.

a t[illegible] string character, / is added and the string is stored unchanged starting left adjusted in the location denoted by ADDR or it is converted according to the special format whose location is denoted by FMT and stored in the locations denoted by VAR1, VAR2, ..., and VARN. This format is a special format different from those in FORTRAN V. I+ is designed so that the teletype user need not memorize or otherwise have at his command, which character positions (analogous to card columns) each number or character string must begin and end in. The format must be a contiguous string of characters, the first one being left adjusted in the location FMT. The only admissible characters are I, R, A, C, D, L and * where:

I means to interpret the next string of characters as an integer,

R means to interpret the next string of characters as a real (single precision floating point),

A means to interpret the next string of characters as fieldata (alpha-numeric not containing a comma),

C means to interpret the next two strings of characters as complex, the first as the real part and the second as the imaginary part,

D means to interpret the next string of characters as double precision floating point,

L means to interpret the next string of characters as a logical,

* means to scan the next string of characters and look up its value in the FORTRAN V constant definitions.

The user may effect the transfer of control associated with a command variable whether by typing only the name of the variable, or by specifying a value for the variable.

Command variable transfer will not occur when there is error in the input string or when the value of the variable is being retrieved.

Whenever an error is detected in the input, the element found to be in error is underlined with the character '*', and the subroutine waits for more input.

All input must be followed by a carriage return.

More specialized teletype I/O may be handled with the two routines described below:

TYPEOUT (ADDR) or TYPEOUT (FMT, VAR1, VAR2, ..., VARN)

Types out on the teletype the fieldata character string which starts left adjusted in the location denoted by ADDR or the string generated from the conversion of the quantities in locations VAR1, VAR2, ..., VARN by means of a FORTRAN V format whose location is denoted by FMT. In either case, the string must be terminated by the terminating character, Δ (11, 7, 8 octal).

TYPIN (ADDR) or TYPIN (FMT, VAR1, VAR2, ..., VARM)

Gets from the teletype the next fieldata character string terminated by a carriage return and line feed character, Δ(0, [illegible] found

return has been typed, then a diagnostic is typed out and the user is expected to retype the line correctly.

Example:

```
      INTEGER DAY,MON,YR,HR,MIN,SEC
      LOGICAL WAIT
      CALL RELOAD
      CALL SWPCHR ('¤ ')
      CALL CHRINT (1,$100)
      CALL TYPOUT ('ENTER DAY, MONTH YEAR¤>Δ')
      WAIT= TRUE.
10    CALL SWAPER(0)
      CALL IDLE
      IF (WAIT) GO TO 10
       .
       .
100   CALL TYPIN ('IAI',DAY,MON,YR)
      CALL CHRINT (1,$200)
      CALL TYPOUT ('ENTER HOUR,MINUTE,SECOND¤>Δ')
110   CALL SWAPER (0)
      CALL IDLE
      IF (WAIT) GO TO 110
      CALL INTRET
200   CALL TYPIN ('III',HR,MIN,SEC)
      WAIT=.FALSE.
      CALL INTRET
```

A comma is used as the string delimiter. If the first character typed in is a comma or if a string of n commas is typed in, then a blank or n-1 blanks (teletype spaces) are assumed respectively. The teletype carriage return and line feed key is used in lieu of a last or terminating comma. The conversion procedure isolates the next string of w non blank characters and converts and stores it according to a FORTRAN format specified by the special format character. If the format character is I, then the FORTRAN V format specified is Iw; R specifies Ew.0; A specifies Aw if w $\leq$ 6 and mA6, An if w > 6 where m = integral part or w/6 and n = w-fm. C specifies $Ew_1.0$, $Ew_2.0$; D specifies Dw.0; L specifies Lw and * specifies one of the above according to the constant t[illegible]. If w = 0, then a zero, blank or false value is stored as requ[illegible]. The first format character specifies that the first character string will be converted as indicated and stored in the location denoted by VAR1, the second in VAR 2, etc.

If the string ABORT is typed in, the program will be aborted through the EXIT subroutine. If another argument, $STMT, is added to the calling sequence, then control will be passed to FORTRAN statement number denoted by STMT when the string [illegible] is typed in. This escape function is useful when inputting variable amount of data in an indefinite loop.

If the number of strings typed in is not equal to the number specified in the call statement, a string is invalid according to the format specified, or the rubout key followed by a carri[illegible]

Display Routines

The IDI display scope permits the user to display vectors and characters on a 1024 x 1024 grid which is about eleven inches square. Horizontal and vertical scaling is provided for both vectors and characters through control knobs on the IDI display console, and allows the user some flexibility in the size and position of his display. The grid origin (0,0) is located in the bottom left corner of screen. Therefore, the range of the grid coordinates are from 0 to 1023 and any value specified outside of this range is treated modulo 1024 causing a "wrap around" effect.

Characters may be written in a large, small, small superscript or small subscript mode. The screen is large enough for 32 lines of 64 characters each when in the large character mode. A large character may be thought of as being in a 16 x 16 box which includes spacing on all sides.

The small character mode permits 128 characters per line but appears to be about 3/4 the height of a large character. Therefore it is probably best to think of the small characters as being in a box 8 x 12.

The following set of routines are designed to permit the user to easily generate a display on the IDI scope. This set is built upon the basic set as described in the Graphics System Technical Report, [1], yet provides nearly all of the flexibility at a great savings in programming effort.

These routines generate display information and store it in a user declared array relative to an index. The array size is limited to 2686 words of memory, and the index always points to the next available location within the array, starting at one. This index may be saved and altered at any point to permit insertions and reconstructions of part or all of the display file. The average user will probably alter the latter part of the display file leaving the first part unchanged since it may contain, for example, information which generates a fixed axis, grid or label.

SETDF (DF, INDEX)

Sets the display file address to the location denoted by the argument DF and the display file index address to that denoted by INDEX. The file and the contents of INDEX are set to unity. All subsequent display routine calls will reference the display file DF and its index INDEX and update the index to the next available location within the display file. This routine must be called at least once for each display file used. A third argument may be added to indicate the size of the display file and is assumed to be 2500 if omitted.

RSETDF (DF, INDEX)

Sets the display file address to DF and the display file index address to INDEX. It does not initialize flags or the variable INDEX. The purpose of this routine is to switch back to a previously used display file DF and/or to declare a previously used or new display file index denoted by INDEX. If a display routine call follows which puts information into the display

fil , it will be inserted relative to the value of INDEX. Again a third argument may be added to indicate the display file size and is assumed to be 2500 if omitted.

ORG(X,Y)

Sets the relative origin to absolute grid coordinates X,Y. All subsequent display routine coordinates will be interpreted relative to X,Y. Initially, X,Y = 0,0.

POS(X,Y)

Position beam at grid coordinates X,Y.

VEC(X,Y)

From current beam position, draw a vector to grid coordinates X,Y.

DOT(X,Y)

Put a dot at grid coordinates X,Y.

DASH(X,Y)

From current beam position, draw a dashed vector to grid coordinates X,Y. The dash is only effective for vectors longer than about 20 grid positions.

Example:

```
      DIMENSION DF(2000)
        .
        .
        .
      CALL SETDF(DF,I)
      CALL ORG(0,100)
      CALL POS(0,0)
      CALL VEC(100,0)
```

```
CALL DOT(100,0)
CALL POS(0,10)
CALL DASH(100,0)
  .
  .
  .
```

LCHAR

Set character size to large

SCHAR

Set character size to small.

SUP

Set character size to small superscript.

SUB

Set character size to small subscript.

WRITAT(X,Y,ADDR) or WRITAT(X,Y,FMT,VAR1, ...,VARN)

Write horizontally starting at relative grid coordinates X,Y the fieldata character string whose first character is left adjusted in location ADDR and whose last character is Δ (1, 7, 8 punch), or write horizonatlly starting at relative grid coordinates X,Y the variables VAR1, VAR2, ..., VARN converted by the format FMT into a fieldata character string whose last character is a Δ.

WRITDN(X,Y,ADDR) or WRITDN(X,Y,FMT,VAR1,VAR2, ...,VARN)

Writes vertically down starting at grid coordinates X,Y in the same manner as WRITAT writes horizontally. Note this sub-

routine uses the margin, return to margin and line feed modes which are also used in the subroutines MARGIN, NXTLIN and SAMLIN described below.

Examples:

```
DIMENSION STRING(2)/'MESSAGE 1Δ'/
DIMENSION FMT1(4)/'(8HMESSAGE ,I2,1HΔ)'/
DIMENSION FMT2(6)/'(8HMESSAGE ,I2,4H OR ,F3.1,1HΔ )'/
  .
  .
  .
I=3
A=4.0
CALL LCHAR
CALL WRITAT(0,500,STRING)
CALL WRITAT(0,400,'MESSAGE 2Δ')
CALL WRITAT(0,300,FMT1,I)
CALL WRITDN(0,1000,'(8HMESSAGE , I2,3H + ,F3.1,1HΔ)',I,A,)
CALL WRITAT(0,200,'X =10Δ')
CALL SUB
CALL WRITAT(0,216,'1Δ')
CALL SUP
CALL WRITAT(0,280,'2Δ')
  .
  .
```

MARGIN(X,Y)

Set the margin at relative grid coordinates X,Y.

NXTLIN( ) or NXTLN(FMT,VAR1,VAR2, ..., VARN)

Return to the margin and write on the next line the character

string at location ADDR or the one generated by the format FMT and the variables VAR1, VAR2, ..., VARN). (See WRITAT).

SAMLIN(ADDR) or SAMLN(FMT,VAR1,VAR2, ..., VARN)

Write a line as does NXTLN except write it at the current line position instead of at the next line position.

Examples:

```
      .
      .
      .
      CALL MARGIN(10,1000)
      DO 10 I=1,50
      CALL LCHAR
      CALL NXTLIN('(3HX =,I3,1HΔ)',I)
      CALL SUB
 10   CALL SAMLIN('(2H  ,I2,1HΔ)',)I)
      .
      .
      .
```

SENDF

If N is the value of the current display file index, then the first N-1 words of the current display file are sent to the PDP-8 and are displayed on the IDI scope.

PLOTDF

If N is the value of the current display file index, then the first N-1 words of the current display file are sent to the CALCOMP incremental plotter and a permanent hard copy corresponding to the display on the IDI scope is produced.

REFERENCES

(1) Copeland, L., Carr, C.S. (1967): Graphics System, Technical Report 4-1, University of Utah Computer Science Department

(2) UNIVAC 1108 EXEC II Programmers Reference Manual (1966): Sperry Rand Corporation, Section 3, page 37.

(3) UNIVAC 1108 FORTRAN V Programmers Reference Manual (1966): Sperry Rand Corporation, Section 7, page 12.