GRAF: Graphic Additions to FORTRAN

by A. Hurwitz and J. P. Citron
IBM Los Angeles Scientific Center
Los Angeles, California

and

J. B. Yeaton*
Health Sciences Computing Facility, UCLA
Los Angeles, California

INTRODUCTION
With the growing widespread use of graphic display devices, higher level graphic display languages which are easy to use are a necessity. Many systems have been developed which are designed around the use of online graphic display terminals. These systems generally have their own language and are designed to run in an environment dedicated to the given system.

On the other hand, means have been developed to allow a higher level language programmer to address graphic devices such as microfilm recorders, incremental potters, or cathode ray tube displays, through the use of pre-packaged subroutines entered by means of CALL statements. This represents no real extension of the language.

GRAF is intended to fill a gap between the two extremes of a package of subroutines in FORTRAN on one hand and a completely dedicated graphical display system on the other. In GRAF, new statements are added to the FORTRAN language. These statements are designed to be as consistent as possible with FORTRAN. The advantages of this design are that GRAF will be easier to implement, easier to teach to FORTRAN programmers, and easier to read.

GRAF was designed with a particular version of FORTRAN and a particular display device in mind. However, the features of GRAF are to a great degree machine independent and modular so that it can be adapted to different configurations.

GRAF was designed to extend OS/360 FORTRAN IV (E Level subset) and to operate with the IBM 2250 Model I display device with a buffer, absolute vector feature, light pen, program function keyboard, and alphameric keyboard.

IMAGE GENERATION

Display variable
The central notion in GRAF is that of a display variable. This is a new type of FORTRAN variable whose value is actually a string of graphic device orders capable of generating directly a display of points, lines, and characters when transmitted to a display device.

Display variables are similar to ordinary FORTRAN variables in many ways. The same rules for naming them apply. They are declared in a declaration statement and can be dimensioned. They can appear in EQUIVALENCE and COMMON statements, and they can be passed to subroutines as arguments. Further, they can be assigned values by an assignment statement similar to the arithmetic assignment statement in FORTRAN.

Notational conventions
In describing the GRAF statements, we will use the following conventions: dv, dv1, dv2,..., dvn will represent any display variables, lower case letters will represent names to be replaced by the programmer—the first letter indicating the type (integer or real) if it is not a display variable, upper case letters will represent names which may not be changed by the programmer. The names “dx” and “df” refer to display expressions and display functions respectively. Items enclosed in brackets are optional and items

*Now with University of California School of Medicine at San Francisco.

From the collection of the Computer History Museum (www.computerhistory.org)
written in a column indicate a choice among those items is to be made.

All functions except display functions follow the usual FORTRAN rules as to type.

**DISPLAY declaration**

Display variables must be declared by using the DISPLAY declaration. Its form is

```
DISPLAY dv1[(k1)], dv2[(k2)], ..., dvn[(kn)]
```

where n is greater than zero and for each dv, j = 1, 2, ..., n dv is declared to be a display variable or an array of display variables whose dimensions are specified by kj, where kj is composed of one to three unsigned integer constants. Each element of an array of display variables can be treated independently just as ordinary subscripted FORTRAN variables can be treated independently of each other.

For example,

```
DISPLAY A, J, Q(17), TR(2, 7, 5)
```

defines A and J to be display variables, and Q and TR to be arrays of display variables.

**Display function**

A display function is a FORTRAN function whose value is a string of graphic orders. The built-in display functions of GRAF are:

- **POINT (x,y)** which generates orders for plotting a point
- **LINE (x,y)** which generates orders for plotting a line
- **PLACE (x,y)** which generates orders to change beam position without plotting
- **CHAR (string, length, mode)** which generates orders for plotting a string of characters
- **PRINT n, list** which creates a string of characters using a list and a format almost exactly like the PRINT of FORTRAN and then generates the orders necessary to plot the resulting string of characters.

The **FORMAT statement** used by the PRINT function has an expanded set of "carriage control characters" which control the size and protection status of characters and insertion of cursors.

**Display expression**

A display expression is a sequence of display variables and display functions separated by plus signs. The value of a display expression is the string of graphic orders which is the concatenation of the values of the display variables and display functions in the order from left to right. Its form is:

```
dv[(k)] = df + dv + df + df + ...
```

where k, if present, specifies the subscript or subscripts of dv.

For example, if A and B are display variables, then:

```
A + POINT(O, YW7) + B
```

is a display expression whose value is the string of graphic orders of A following by the orders generated by POINT followed by the orders generated by B.

**Display assignment statement**

The display assignment statement is used to assign a value to a display variable. Its form is:

```
dv [(k)] = df + df + df + ...
```

where k, if present, specifies the subscript or subscripts of dv.

For example, the following are display assignment statements, assuming the variables A, B, SQU, POLE, K99 appeared in the DISPLAY statement below:

```
DISPLAY A, B, SQU, POLE(11), K99(3, 2, 4)
A = B
A = A + B
A = B + A
SQU = POLE(5) + POLE(3)
POLE(1) = PLACE(RX, 0)
POLE(1) = PLACE(RX, 0) + LINE(X3, Y7)
K99 = PLACE(0, 0) + PRINT 14, (ZK(I), I = 1, 7) + PLACE(2000, 2000)
```

**Coordinate specification**

Subroutines are provided to specify a coordinate transformation so that the built-in functions PLACE, POINT, and LINE which use x, y coordinates will be able to make the transformation from user coordinates to device coordinates. If no specification is made, device coordinates will be used throughout.

**Resetting a display variable**

In order to set a display variable to the empty string, the **RESET subroutine** is used. Its form is:

```
CALL RESET (dv1, dv2, ..., dvn)
```

**OUTPUT OF A DISPLAY VARIABLE**

The transmission of display variables to the display device and the subsequent erasure and reploting are controlled by the FORTRAN functions PLOT, UNPLOT, ERASE, and the subroutine BLANK.

**PLOT**

The form of the PLOT function is:

```
PLOT (dv1, dv2, ..., dvn)
```

The function transmits the values of its arguments, that is, the strings of graphic orders corresponding to the display variables, to the device buffer if there is enough storage available in the buffer. If not enough storage is available for all the strings then no transmission takes place.

The numerical value returned by PLOT is positive if transmission took place and negative otherwise; its absolute value is equal to the amount of available buffer storage.
If any of the display variables are dimensioned and are not subscripted, then the entire array is plotted. Each time a display variable is plotted, a new instance is said to be created for that display variable. The newly created instance becomes the current instance. As many instances as desired may be created for a display variable and the value of the variable may be changed between the uses of PLOT. For example, the following sequence of statements creates three instances of A, two of which have the same value:

\[
A = \text{PLACE}(0, 0) + \text{LINE}(0, 4095) \\
T = \text{PLOT}(A, A) \\
A = \text{PRINT} 18, U, V, W \\
T = \text{PLOT}(A)
\]

UNPLOT

The FORTRAN function UNPLOT does exactly the opposite of PLOT. It removes the current instance (if there is one) of a display variable and makes the previous current instance (if there is one) the current instance. Its form is:

\[
\text{UNPLOT}(d_v, d_{v2}, \ldots, d_{vn})
\]

The numerical value returned is the amount of available buffer storage after its execution.

ERASE

The ERASE function is the same as UNPLOT except that all instances of the display variables appearing as arguments are removed.

BLANK

The subroutine BLANK performs an ERASE on all active display variables, that is, it blanks the display device.

For example, if A had been plotted as in the above example, the statement

\[
TX = \text{UNPLOT}(A)
\]

would remove the current instance of A, that is, the characters generated by the PRINT. The statement

\[
TX = \text{UNPLOT}(A, A) \text{ or } \text{UNPLOT}(A) + \text{UNPLOT}(A)
\]

would remove two instances of A, and

\[
TX = \text{ERASE}(A)
\]

would remove all instances of A.

READING ALPHAMERIC INPUT

At this point, we shall assume that the display device being used is an IBM 2250 with a buffer. In order to type alphameric input into a 2250 buffer, a cursor must be set in the buffer. GRAF provides a means of setting such a cursor in a display variable. When the display variable is plotted the cursor will be set in the buffer and be displayed on the screen. The characters which are typed on the keyboard replace the characters that were in the buffer and were being displayed. The new characters are then displayed on the screen.

READ

The READ statement is provided to read the contents of the display buffer into main storage. Its form is

\[
\text{READ}(dv, \text{format-number}) \text{ list}
\]

If the display variable dv has an instance displayed on the screen, then all the character information from the current instance is read from the buffer into the display variable in main storage, converted according to the format specified, and sent to the variables in the list. If no instance was on the screen, the buffer is not read but the conversion is carried out with the contents of the display variable as it is.

Setting and removing cursors

A cursor can be set in a display variable either by a PRINT statement using the correct control character in its format or by the SETCUR subroutine. A cursor in a display variable is removed by the subroutine RMVCUR or by assigning a new value to the display variable by an assignment statement or by calling RESET.

There can be only one cursor on the screen at one time. The cursor is removed from the buffer if it is in an area of the buffer corresponding to an instance of a variable which has been removed by UNPLOT or ERASE. Also, a call to BLANK will remove the cursor.

ATTENTION HANDLING

In provisions for attention handling, GRAF shows most clearly the conditions under which the system was designed and developed. GRAF was to be used with Operating System 360 FORTRAN (E level) and to be implemented using Operating System 360 Express Graphic Support. The Express Graphic Support does not inform the user’s FORTRAN program that an attention has occurred unless the FORTRAN program explicitly requests such information. Therefore, GRAF does not provide an elaborate method of handling attentions.

DETECT

The DETECT FORTRAN function returns information about the occurrence of attentions generated by the light pen, programmed function keyboard, end-of-order sequence, alphameric end key, and alphameric cancel key. The value of the function DETECT indicates the type of attention. Its form is:

\[
\text{DETECT}(\text{inqarray})
\]

where inqarray is an array of dimension 5. DETECT returns information as follows:

\[
\text{if } \text{NDET} \text{ is an array of dimension 5 and } J = \text{DETECT}(\text{NDET})
\]
were executed, the possible results are shown in the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>J = 0</td>
<td>no attention had occurred.</td>
</tr>
<tr>
<td>J = -1</td>
<td>a light pen attention occurred and NDET (1) = a number corresponding to the display variable causing the detect and then NDET (1) &gt; 0 implies the current instance was detected or NDET (1) &lt; 0 implies the current instance was not detected</td>
</tr>
<tr>
<td>J = 1</td>
<td>a programmed function key attention occurred and NDET (4) = key number</td>
</tr>
<tr>
<td>J = 2, 3, or 4</td>
<td>an attention occurred corresponding to the end key, cancel key, or end-of-order sequence respectively and nothing is returned to NDET.</td>
</tr>
</tbody>
</table>

**DETAIN**

DETAIN works exactly the same as DETECT except that it waits until an attention occurs before returning, hence it never returns the value zero. DETAIN also allows the system to execute other tasks while the user program is waiting for a detect.

**LPNAME**

The value of the name of the display variable, the first element of the array inqarray (NDET (1) in the above example), is of little use without the LPNAME function which analyzes its meaning. The form of LPNAME is:

```
LPNAME(dvname, d1, d2, ..., dn)
```

The value of LPNAME is 0, 1, 2, ..., or, n. In using LPNAME, the variable dvname has a numerical value corresponding to some display variable, that is, a value returned to the first element of the array argument of a DETECT or DETAIN after a light pen detect. LPNAME determines whether or not that numerical value corresponds to any of the display variables dv1, dv2, ..., dvn. If a correspondence is found with say dvj when the value returned by LPNAME is j. If no correspondence is found, then the value of LPNAME is zero.

The arguments of LPNAME can be dimensioned display variables. In that case, if dvname matches any element of the array the value of LPNAME will show a match with that array. In addition, if dv is an array, if the notation dv/i1/i2/i3 is used, and if dvname matches an element of the three dimension array dv, then the subscripts where the match occurred will be returned into the integer variables i1, i2, i3. Similarly one can use the notation with two and one dimensional arrays.

For example, if the following statements occurred in a program:

```
DIMENSION INQ (5)
DISPLAY S, T, U(S, 4), W(7, 3, 5)
J = DETAIN (INQ)
```

and a light pen detect occurred, then

- if T caused the detect
  
  \[ \text{LPNAME (INQ(1), S, T, U, W)} = 2, \text{ or} \]

- if U (2, 3) caused the detect
  
  \[ \text{LPNAME (INQ(1), S, T, U, W)} = 3 \]

The use of the name return from DETECT and DETAIN relieves the programmer of keeping track of his display by a separate bookkeeping scheme such as building a table of internal and external names.

In general, all the things which need to be plotted, erased, or detected with the light pen as distinct entities will be defined using distinct display variables or distinct elements of an array of display variables.

**Miscellaneous attention functions**

Other functions are provided to enable and disable attentions, and to turn the indicator lights on the programmed function keyboard on and off.

**USER WRITTEN DISPLAY FUNCTION**

The built-in display functions are PLACE, POINT, LINE, PRINT, and CHAR. The FORTRAN programmer can use GRAF to create display functions of his own which can be used in display assignment statements. To do this, he compiles a program in a way similar to compiling an ordinary FORTRAN function program. As the first statement of such a display function program, the statement DISPLAY FUNCTION is used. Its form is:

```
DISPLAY FUNCTION functionname (arg1, arg2, ..., argn)
```

From the collection of the Computer History Museum (www.computerhistory.org)
The rest of the program is written like a FORTRAN function program, but including GRAF statements.

For example, the programmer could create and use a display function called BOX as follows:

```fortran
DISPLAY FUNCTION BOX (X, Y, U, V)
BOX = PLACE (X, Y) + LINE (U, Y) + LINE (U, V) + LINE (X, Y)
RETURN
END
```

The programmer can now use the display function BOX in a manner similar to the built-in functions PLACE, POINT, and LINE. For example:

```fortran
A = BOX (0, 0, 2000, 2000) + BOX (3000, 2000, 4000, 4000).
```

SUMMARY

We have defined a new type of variable, a display variable, in GRAF and have tried to extend FORTRAN in a consistent fashion to deal with the display variables. We have tried to minimize the number of new statements a programmer will have to learn in order to use a display device and at the same time, define a system that will be powerful enough to enable him to use all the capabilities of the display device. Further, we feel that coding, debugging and simply understanding the logic of a program from its listing are all made much easier by avoiding CALL statements with long argument lists for frequently needed graphic routines.

ACKNOWLEDGMENT

The authors would like to acknowledge the many useful discussions they have had with Dr. D. G. Cantor and Robert B. Keller.

REFERENCES

1. IBM Operating System/360 FORTRAN IV (E Level Subset)
   Form C28-6513
2. IBM System/360 Component Description
3. IBM Operating System/360 Graphic Programming Services

From the collection of the Computer History Museum (www.computerhistory.org)