A Device Independent Input Structure for a High Level Graphics Language

C.D.O'Brien and H.G.Bown

Communications Research Centre
Department of Communications
Ottawa, Ontario

ABSTRACT

The syntax of an interactive graphics language 'IMAGE', being developed at the Communications Research Centre, will be presented with particular emphasis on the features of the language which make it hardware input device independent. Six forms of interaction have been identified and facilities are provided in the language with which to program for six virtual input devices which ideally suit these forms of interaction. The programmer writes his programs in a device independent manner by referencing only the virtual device. The syntax for device independent identification removes the graphics programmer from interrupt level programming and structures his interaction dialogue. A number of example 'IMAGE' programs are given to illustrate the simplicity, power and device independence of the input structure and syntax.

ABRÉGÉ

La syntaxe d'un langage graphique interactif "IMAGE", en cours de développement au Centre de recherches sur les communications, sera présentée en mettant un accent particulier sur les caractéristiques du langage qui le rendent non tributaire du dispositif d'entrée. Six formes d'interaction ont été identifiées et le langage permet d'établir le programme en fonction de six organes d'entrée virtuels qui conviennent de façon idéale à ces formes d'interaction. Le programmeur écrit son programme sans tenir compte du type d'appareil, en se référant uniquement à l'organe virtuel. La syntaxe d'identification non tributaire du type d'appareil dispense le programmeur graphique du soin de programmer les interruptions et structure son dialogue d'interaction. Un certain nombre de programmes "IMAGE" sont donnés pour illustrer la simplicité et les possibilités de la structure d'entrée et de la syntaxe et pour démontrer qu'elles ne sont pas tributaires du type d'appareil utilisé.

INTRODUCTION

Interaction with a computer graphics display can be attained through the use of many input devices such as tablets, light-pens, knobs, switches, pushbuttons and keyboards, but whatever the device, there are only a few basic modes of interaction. A programmer who makes use of a particular device in an implementation dependent manner restricts his program to operating only on a small number of machines. Potentially his program could execute on any machine having devices which support the mode of interaction by which his program communicates. The cost of software is high so it is economically advantageous to write portable software.

This paper considers the problem of providing graphics software portability by the use of a specialized high level graphics language with a device independent input structure. The arithmetic, character string manipulative, and logical algorithims provided in most high level languages are usually portable because the syntax allows for their machine independent definition. However, the facilities provided within most high level languages for graphics programming reference the graphic I/O devices in a dependent manner. For example, some FORTRAN graphic subroutine packages [1] assume that the 'Display Processor Unit' references an in-core display file. Thus, there appears to be a requirement for a device independent input structure within a programming system that permits input/output devices to be functionally referenced. The 'IMAGE' language [2] being developed at the Communications Research Centre provides such a facility.

THE IMAGE LANGUAGE

A high level graphics language named 'IMAGE' has been designed by the authors in order to provide a graphics application programmer with the ability to easily program interaction. It utilizes the better features of several current graphics languages and combines these features with a unique interaction control structure. This control structure, the display picture description syntax and the hardware independent handling of input devices, are the main features of the language. The device independent input/output structure permits the implementation of a portable language syntax, since there are no references to particular display hardware devices.

The syntax for device independent identification is of particular intrest because of its unusual form. All displayed information is delimited into graphical OBJECT blocks. An ACTION block may be associated with a block of OBJECT code to indicate what action code will be executed upon an identifier strike on that particular object. Thus, an ACTION block is basically a high level interrupt handler routine. A program is written as a group of OBJECT and ACTION block pairs with each action associated with an object. This removes the graphics programmer from low level interrupt programming and structures his interaction dialogue. Other methods of providing for device identification utilize a polling mechanism. For example, most FORTRAN graphics subroutine packages only provide a routine to request the activity of a particular device. The advantage of an interrupt driven identification scheme is that it encourages the writing of natural man-machine dialogues. Therefore, an IMAGE program reads as a series of OBJECT /ACTION pairs whose execution is interactively controlled.

FORMS OF INTERACTION

Conceptually, the 'IMAGE' language recognizes six input functions and associates them with six virtual input devices. The language provides facilities ideally suited for programming interaction with these virtual input devices. The programmer writes his program in a device independent manner by referencing the virtual devices utilizing the IMAGE instructions associated with the six input functions. A real hardware device may be well suited to perform one class of interaction or it may be able to handle several classes of interaction with varying ease. System software is used to permit different real hardware devices to emulate the functions of the virtual device.

Three of the six input functions are general in nature and three relate specifically to the graphics display. The specific graphical functions and their associated default virtual devices are:

	SPECIFIC FUNCTION	VIRTUAL DEVICE
I.	Identifying	PICKER
II.	Sketching	DIGITIZER
III.	Positioning	LOCATOR

I. IDENTIFYING

The function of identifying is handled through the interrupt driven OBJECT / ACTION structure of IMAGE. The ACTION routine associated with an OBJECT is executed when an identifier interrupt associated with that OBJECT occurs. During the execution of the ACTION associated with the OBJECT identified, the reserved integer variable 'ITEM' contains the sub-delimiting tag count, indicating which part of the object was identified. The x,y co-ordinates at which the identifier stylus struck the displayed OBJECT may be obtained using the IMAGE command 'COORD[INATES] (xvar,yvar)', where xvar and yvar contain the requested information. The virtual device associated with the function of identifying is assigned using the following instruction:

```
                             [ PICKER ]
    IDENTIFY [ USING ]   :   [ DIGITIZER ]
                             [ LOCATOR ]
```

The virtual device PICKER may correspond to a light-pen, the DIGITIZER to a tablet and the LOCATOR to a device such as a track-ball or joy stick. By using suitable software techniques in the run-time system any of these real devices can perform the task of IDENTIFYing. What these instructions achieve is to separate the task of IDENTIFYing from the devices which perform it.

II. SKETCHING

The sketching facility within IMAGE provides a mechanism whereby the absolute x,y co-ordinates visited by the drawing stylus are accumulated. The primary use of this facility is in the creation of free-hand 'inked' drawings on the display surface. The form of visual feedback is under direct program control. The following commands permit control of the sketching mechainism:

SKETCH [ON]   or   SKETCH OFF

- enables or disables the sketching mechanism.

SKETCH RESOL[UTION] (factor)

- defines the zone of insensitivity about the stylus. An x,y position is accepted as valid only if it differs from the last point by an amount specified by 'factor'.

SKETCH Q[UEUE] (length)

- a definitional command used at the beginning of a program to define the length of the queue for accumulated points. The positions of all points visited by the drawing stylus are stored in this queue until they are requested by the 'SKETCH LOCATION' command.

SKETCH LOC[ATION] (xvar,yvar)

- obtains the x,y drawing stylus positions from the queue.

SKETCH Q[UEUE] CLEAR

- this command clears the sketch queue.

III. POSITIONING

The positioning facility permits specific screen locations to be indicated by utilizing a controllable marker. A single marker is available on the screen and the following commands modify its operation:

MARKER AT(x,y) - sets the marker to position x,y.

MARKER ; OFF & MARKER ; ON - enables & disables the marker.

MARKER ; VERT[ICAL] or
MARKER ; HOR[IZONTAL] or
MARKER ; FIXED or
MARKER ; SLOPED (angle) - constrains marker motion.

MARKER LOC[ATION] (xvar,yvar) - obtains the current x,y position of the marker.

The three general input functions and their associated default devices are:

	SPECIFIC FUNCTION	VIRTUAL DEVICE
	========	=======
IV.	Input of textual strings	KEYBOARD
V.	Pushbuttons or switches	PUSHBUTTONS
VI.	Input a numeric value	VALUATOR

IV. INPUT OF TEXTUAL STRINGS

Although IMAGE provides record-oriented input in the conventional manner via an INPUT statement similar to a FORTRAN 'READ' statement, it also provides a unique interrupt based facility. The KEYBOARD command which provides this capability is a special case of the OBJECT block and is described below:

KEYBOARD or
KEYBOARD (activation character) ,... or
KEYBOARD CHAR[ACTER]

- enables interrupts from the keyboard. When a character is typed on the keyboard it becomes the current character in the special reserved string buffer KEYBUF. The Carriage Return and Line Feed characters are the default activation characters but other or all characters may be specified. If an activation character is typed, the ACTION associated with the OBJECT containing the keyboard statement is executed.

V. PUSHBUTTONS

The operation of a pushbutton is similar to the operation of an OBJECT block within the identifier structure of the program. A pushbutton is considered as a special type of object within an OBJECT block in a similar manner to the KEYBOARD statement. Upon a pushbutton strike, the action associated with the OBJECT statement is executed and the reserve variable 'ITEM' contains the number of the pushbutton selected. The format of the pushbutton statement is:

PUSHBUTTON[S] (number, ...)

- enables pushbutton interrupts from the indicated pushbutton number.

VI. VALUATORS

The input of numeric information through a keyboard requires syntax checking for the valid specification of the number. It is desireable to have a direct form of numeric input as can be provided by a hardware device such as a potentiometer. Such a virtual device is termed a valuator [3]. As the valuator is adjusted, the value is updated continuously and returned via interrupt to an ACTION block. A valuator is also treated as a special block within an OBJECT block. The ACTION routine associated with the special object VALUATOR is executed if a resolvable change in the value associated with the valuator device is observed. The valuator command has the form:

VALUATOR [# num] (var)

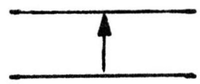
- enables the valuator device of number 'num'. The value associated with the valuator will be stored in variable 'var'. and will always be in the range -1 to 1.

EXAMPLE IMAGE PROGRAMS

The following IMAGE programs illustrate the use of the three specific device independent functions: identifying, sketching and positioning.

IDENTIFYING

The following example uses both the TAG sub-delimiting feature and the ability to obtain the x,y co-ordinate 'struck' with the stylus. The program draws two parallel horizontal lines on the screen. Upon a strike with the identifier on either of the lines, a vertical arrow is drawn between the two lines to mark the indicated spot. The arrow points up or down depending on which line was indicated. The diagram below shows how the screen would look after the upper line was struck in the middle.



```
INTEGER X
OBJECT
    LINE 500,0 AT(100,200)          ** DRAW THE FIRST LINE
    TAG                             ** DELIMIT WITH A TAG
    LINE 500,0 AT(100,100)          ** DRAW THE SECOND LINE
ACTION
    REMOVE                          ** ERASE ALL MATERIAL
*                                   **   DRAWN BY PAST ACTIONS
    COORD (X,)                      ** GET THE X POS. INDICATED
    IF (ITEM = 1)                   ** UPPER LINE ?
       DO ARROW AT(X,100)           ** YES, DRAW ARROW UP
    ELSE                            ** NO, THEN LOWER LINE.
       DO ARROW ;ROT(180);AT(X,200) ** DRAW ARROW DOWN
    FIN
*
PROCEDURE ARROW
    LINE 0,100 AT(0,0)              ** DRAW THE VERTICAL SHAFT
    LINE -10,10/-10,-10 AT(10,90) ** DRAW THE ARROW HEAD
END
```

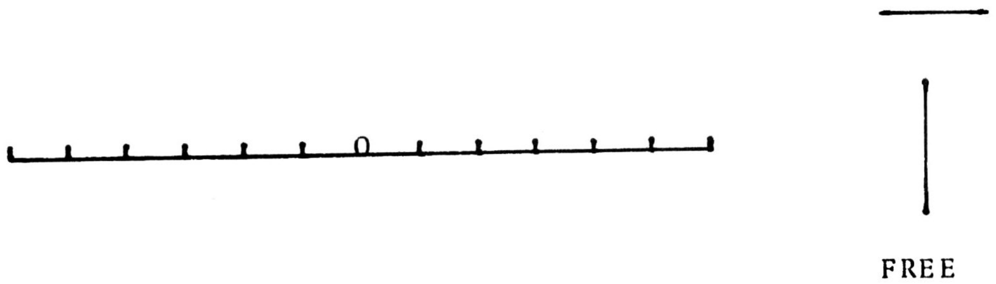
SKETCHING

The following program illustrates the use of the IMAGE sketching commands. The purpose of this example is to allow sketching in a free-hand manner on the display screen. A line will appear on the screen joining the points visited by the stylus. A coarse resolution of 20 screen co-ordinates on a 1000 unit square screen was chosen so that the number of data points and the rate at which they are collected is not too great. Straight lines are drawn between the points to provide a continuous curve. In order to keep the program simple no facility has been provided to allow 'lifting' the stylus. Only one continous line may be sketched. The sketching is enabled or restarted by touching the light-button 'SKETCH' . The light-button 'STOP' halts execution.

```
INTEGER X1,X2,Y1,Y2
SKETCH QUEUE (20)    ** DEFINE THE LENGTH OF THE SKETCH QUEUE
*
ENTRY
   IDENTIFY USING : PICKER   ** USE PICKER DEVICE TO IDENTIFY
   SKETCH USING : DIGITIZER ** USE DIGITIZER DEVICE TO SKETCH
   SKETCH RESOLUTION (20)
*
OBJECT
   TEXT 'SKETCH' AT(900,700)
ACTION
   SEEK                ** ENABLE THE IDENTIFIER INTERRUPTS SO
*                      ** THIS ACTION ROUTINE MAY BE ABORTED.
   SKETCH Q CLEAR      ** REMOVE PREVIOUS SKETCHES SO THAT
   REMOVE              **  SKETCHING MAY BE RESTARTED.
   SKETCH ON           ** ENABLE THE SKETCHING MECHANISM.
   SKETCH LOC(X1,Y1)   ** GET THE FIRST POINT.
   REPEAT
      SKETCH LOC(X2,Y2)   ** GET THE NEXT POINT.
      LINE TO X2,Y2 FROM X1,Y1     ** JOIN WITH A LINE
      LET X1 = X2                  ** SAVE THIS POINT FOR
      LET Y1 = Y2                  **    REFERENCE.
   FIN
*
OBJECT
   TEXT 'STOP' AT(900,640)
ACTION
   STOP                    **  TERMINATE EXECUTION
*
END
```

POSITIONING

The following program allows the use of the marker to do constrained drawing. Only horizontal or vertical lines may be drawn. In this program, the virtual device PICKER has been assigned the job of positioning in order to indicate the syntax of such an assignment. When this program is executed the marker appears ON and FREE in the default position in the centre of the screen, and it may be freely positioned to anywhere on the screen. The first menu item is a horizontal line. A strike on this OBJECT constrains motion to a horizontal direction. The character '0' is placed on the screen to indicate the current marker position, and a scale is drawn on the screen along the axis of allowed motion. The marker may be moved left or right along this scale. If either the vertical line or 'FREE' light-button is 'struck', a line is drawn from the position of the marked character '0' to the current marker position, the scale is erased, and the task of the selected light-button is performed. The vertical line light-button allows vertical lines to be drawn in the same manner in which the horizontal line light-button allows the marker to be placed anywhere on the screen. The screen would appear as below after the horizontal light-button was struck.




```
INTEGER X,Y,XO,YO,FLAG
ENTRY
   LET FLAG = 0.0              ** INDICATE THE MARKER IS FREE
   POSITION USING : PICKER     ** USE PICKER DEV. TO POSITION
   DISPLAY EXCLUDING SCALE     ** EXECUTE ALL OBJECT BLOCKS
*                              **  EXCEPT OBJECT NAMED 'SCALE'
OBJECT
   LINE 100,0 AT(900,700)      ** HORIZONTAL LIGHT-BUTTON
ACTION
   IF (FLAG = 0)               ** IF THE MARKER IS FREE
      MARKER LOC (XO,YO)       ** GET THE CURRENT MARKER POS.
   ORIF (FLAG = 2)             ** IF THE MARKER IS VERTICAL
      ERASE SCALE              ** ERASE THE OLD SCALE
      MARKER LOC (X,Y)         ** GET THE CURRENT MARKER POS.
      LINE TO X,Y FROM XO,YO ** DRAW A LINE FROM THE LAST
*                              ** TO THE CURRENT MARKER POS.
      LET XO = X               ** SAVE THE CURRENT MARKER POS.
      LET YO = Y
   FIN                         ** IF THE MARKER IS CONSTRAINED
*                              **   TO HORIZONTAL, NO CHANGE.
   LET FLAG = 1                ** INDICATE HORIZONTAL
   MARKER HORIZONTAL           ** CONSTRAIN MARKER TO HORIZ.
      DISPLAY SCALE            ** DISPLAY THE SCALE ABOUT XO,YO
*
OBJECT
   LINE 0,100 AT(950,550)      ** VERTICAL LIGHT-BUTTON
ACTION
   IF (FLAG = 0 )              ** IF THE MARKER IS FREE
      MARKER LOC (XO,YO)       ** GET THE CURRENT MARKER POS.
   ORIF (FLAG = 1)             ** IF THE MARKER IS HORIZONTAL
      ERASE SCALE              ** ERASE THE OLD SCALE
      MARKER LOC (X,Y)         ** GET THE CURRENT MARKER POS.
      LINE TO X,Y FROM XO,YO ** DRAW A LINE FROM THE LAST
*                              ** TO THE CURRENT MARKER POS.
      LET XO = X               ** SAVE THE CURRENT MARKER POS.
      LET YO = Y
   FIN                         ** IF THE MARKER IS CONSTRAINED
*                              **   TO VERTICAL ; NO CHANGE
   LET FLAG = 2                ** INDICATE VERTICAL
   MARKER VERTICAL             ** CONSTRAIN MARKER TO VERT.
      DISPLAY SCALE            ** DISPLAY SCALE ABOUT XO,YO
*
OBJECT
   TEXT 'FREE' AT(900,400)     ** 'FREE' LIGHT-BUTTON
ACTION
```

```
   ERASE SCALE                  ** ERASE THE DRAWING SCALE
   MARKER LOC (X,Y)             ** GET THE CURRENT MARKER POS.
   LINE TO X,Y FROM XO,YO       ** DRAW A LINE FROM THE LAST POS.
*                               **  TO THE CURRENT MARKER POS.
   LET FLAG = 0                 ** INDICATE THE MARKER IS FREE
   MARKER ON                    ** ENABLE THE MARKER WITH NO
*                               **  RESTRICTIONS.
OBJECT SCALE                    ** OBJECT TO DRAW A SCALE
   IF (FLAG = 1)                ** HORIZONTAL SCALE
      REPEAT X = 100,800,100 ** DRAW  TICKS 100 UNITS APART
         LET Y = Y + 10         ** TOP OF TICK
         LINE 0,-10/100,0 AT(X,Y) ** DRAW A TICK & LINE SECT.
      FIN
      LINE 0,10                 ** DRAW THE LAST TICK
   ORIF (FLAG = 2)              ** VERTICAL SCALE
      REPEAT Y = 100,800,100 ** VERT. TICKS 100 UNITS APART
         LET X = X + 10         ** TOP OF TICK
         LINE 0,-10/100,0 AT(X,Y) ** DRAW A TICK & LINE SECT.
      FIN
      LINE 10,0                 ** DRAW THE LAST TICK
   FIN
   TEXT 'O' AT(XO,YO)           ** MARK THE CURRENT POINT
END
```

SUMMARY

IMAGE is an interraction oriented interrupt based language which provides the application programmer with a tool for writing natural interaction dialogues. The primary emphasis of the design is on solving the man-machine interaction problem. The OBJECT / ACTION structure and the interaction control mechanism supplied, provide a powerful and easy-to-use tool for solving this interaction problem. The LIGHTBUTTON structure in Maclean's 'ICPL' [4] and the display procedure structure of Newman [5] were assimilated with a conceptulization of the manner in which a man interacts with a machine based on work by Foley and Wallace [3]. This produces a device independent method of controlling man-machine interactions. The procedure and function oriented approaches of EULER/G [6] and GRAPPLE [7] were combined with structured programming concepts to produce an easy to use picture description grammar. Graphical input response facilities are provided in a device independent manner through the use of six virtual input devices. A SKETCH and POSITION mechanism allows x,y co-ordinates to be input, while an IDENTIFYing mechanism based on the OBJECT / ACTION structure provides a powerful interaction mechanism. A KEYBOARD and a PUSHBUTTON mechanism provide interrupt based character and control input and a VALUATOR [3] allows input of data in numeric form.

REFERENCES

1) "GRAPHIC-15 Programming Manual", DEC-15-ZFSA-D, Digital Equipment Co., Maynard Mass.

2) O'Brien,C.D., "IMAGE - a language for the Interactive Manipulation of a Graphics Environment", M.Eng. Thesis, Carleton University, Ottawa, Canada, 1975.

3) Foley,J.D., and Wallace,V.A., "The Art of Natural Graphic Man-Machine Communication", Proc. IEEE, Vol.62, No.4, April, 1974.

4) Maclean,M.A., "Designing a Language for Interactive Control Programs", 2nd Man-Computer Communication Seminar, 31 May - 1 June, 1971.

5) Newman,W.M., "Display Procedures", CACM, Vol.14, No.11, Oct., 1971.

6) Newman,W.M., Gourand,H. and Oestreicher,D.R., "A Programmer's Guide to PDP-10 EULER", Univ. of Utah, Report No. UTEC-CSc-70-105, June, 1970.

7) "GRAPPLE Language Reference Manual", Bell Northern Research, Edition 4.0, Sept. 1973.