PICADE
PROMPT INTERACTIVE CREATION OF ACTIVE DISPLAY ENVIRONMENTS

R.D. Trivett
Carleton University

ABSTRACT

Techniques developed at Carleton University as a basis for interactive graphics program development are presented. These techniques are premised on the objective of creating an environment that facilitates the task of programming graphics applications. Three distinct sections describing the program packages are included. The first specifies a set of subroutines used for the programming of graphics interrupt handlers in FORTRAN. The second section describes a system which controls the interactive creation of display images. This technique can be interfaced with the first to produce the PICADE system which facilitates the generation of graphics application programs. The last section of the paper describes an approach used to cause lines on the screen to appear to flow in a defined direction. The integration of support for this technique into the basic graphic's system programs is described. This technique can be used for the animation of system or computer flow diagrams.

RESUMÉ

On décrit des techniques développées à l'université de Carleton qui sont à la base de développement de programmes de la representation graphique interactive. On a présupposé que le but est de créer une ambiance appropriée au travail de la programmation des applications graphiques. Il y a trois sections distinctes dans la communication. Dans la première on décrit une série de sous-programmes utilisée pour programmer en FORTRAN le maniement des signaux d'interruption; dans la deuxième section on décrit un système de programmes pour dessiner et représenter des images selon la méthode interactive. On peut employer conjointement les deux techniques pour former le système PICADE qui facilite la création de programmes graphiques d'application. Dans la dernière section, on décrit la programmation pour que les lignes se déplacent sur l'écran dans une direction définie. On décrit aussi la combinaison de cette programmation avec le système de base de programmes graphiques. On peut utiliser la programmation pour dessiner les organigrammes et les ordinogrammes.

PICADE
PROMPT INTERACTIVE CREATION OF ACTIVE DISPLAY ENVIRONMENTS

R.D. Trivett
Carleton University

Introduction

This paper describes techniques developed at Carleton University as a basis for interactive graphics program development. The paper is divided into two parts. Part A discusses basic graphics techniques and contains two sections; 1) a technique for handling graphic's interrupts in FORTRAN, and 2) a system for the interactive creation of display images. Part A concludes with a description of the integration of these two techniques into a system (given the acronym PICADE) which permits the Prompt Interactive Creation of Active Display Environments. Part B discusses a technique for the generation of flowing lines on the display. This could be used as part of a package for animating system or computer program flow diagrams.

The development was performed on a PDP-15 computer with an associated Graphics-15 processor. These units operate in a dual processor environment both sharing the 24K of 18 bit core memory. The basic mechanism for creating displays is one that uses the PDP-15 processor to run a program which generates the required display file code and stores it in a reserved data area in memory. When the display file is complete the PDP-15 executes an instruction which commands the display processor (in the Graphics-15) to start executing the display file instructions. By executing these instructions the graphics processor performs the corresponding display controlling functions and, vectors and characters are drawn on the cathode ray tube as specified in the display file. The display processor has, included in its basic instruction set, a jump or branch instruction. The last instruction of a display file is one which transfers control back to the beginning. Thus, a display file will be repeatedly executed by the display processor. It is this cyclic re-execution of the display file which causes the display to be refreshed on the cathode ray tube.

Part A - Basic Graphics Techniques

1. Graphics Interrupt Handling in FORTRAN: An interactive environment is created by the provision of a lightpen and/or pushbuttons at the display console. These are used for communication between the operator and a running program. Activation of either mechanism results in a "program interrupt". Normally a program is executed sequentially from memory until a branch or jump instruction is encountered. The "flow of control" is pre-programmed and can aptly be described as "linear". When an interrupt occurs the currently executing sequence of instructions is abandoned and a special interrupt handling sequence is entered. When this sequence is completed the original program can be re-started at the point of interruption or control can be transferred to a completely new sequence of code. The programming of interrupts can be described as "non-linear". This section describes a method for programming the handling of graphic's interrupts in FORTRAN. Two factors contributed to the decision to develop this technique:

i) a FORTRAN compiler existed on the PDP-15 computer, as it does on most computers, and thus a fast, efficient compilation of the resulting programs would be possible,

ii) FORTRAN is a widely known language, and the effort required to learn how to apply the graphics extensions would be minimal.

These factors in the light of limited programming and computer resources were sufficient to determine the decision.

It is required that certain operations be performed when either of the interrupt mechanisms is initiated. This occurs when a pushbutton is depressed on the console, or the lightpen is pointed (with the shutter open) at a section of the display that is defined as lightpen sensitive. As the hardware detects a graphic's interrupt, the graphics processor suspends execution of the display file. An assembly language program extracts several parameters from the display processor, re-starts execution of the display file and, when this technique is enabled, transfers control to the appropriate FORTRAN routine. Because the graphics processor has re-started execution of the display file another interrupt could be generated before completion of the handling in FORTRAN of the first interrupt. Reentry to the FORTRAN handler is blocked, by software, until a "return from interrupt" sequence is entered from FORTRAN. This prevents multiple entries into the FORTRAN program section and thus preserves the integrity of the variables local to the interrupt handler. The parameters stored by the assembly language sequence which define the nature of the graphic's interrupt are:

i) the name register (NREG),

ii) the graphics processor program counter (INTPC),

iii) the X and Y position of the lightpen on lightpen generated interrupts (INTX, INTY),

iv) the pushbutton number on a pushbutton generated interrupt (INTPB).

The capitalized words bracketed with each item in the above list are the FORTRAN variable names used to store the corresponding parameter values. The use of these variable names is clarified in the description of the system's subroutines included below.

The "name register" is a register in the graphics processor which can be set to contain different values under graphic's program ontrol. The name register can be loaded with a unique number before each different section of lightpen sensitive display file code is executed. The number in this register identifies the source of a lightpen interrupt and can be used by the interrupt handling sequence. In the Graphics-15 it is a 7 bit register and, therefore, 128 different "names" or register values are possible.

The "graphics processor program counter" is a register that stores the core memory address of the instruction in the display file being executed by the graphics processor when the interrupt occured. This parameter can also be used to identify the source of lightpen interrupts.

The "X and Y position" registers indicate the source of the lightpen interrupt and are referenced when the physical screen position of the lightpen is required. In some graphics systems this is the only information available to the programmer after a lightpen interrupt has occured.

The "pushbutton number" is of abvious value when processing a pushbutton interrupt. There are six pushbuttons on the Graphics-15 console and various program determined functions can be associated with these.

Several subroutines are used to set up and control the handling of interrupts in the FORTRAN environment. Table 1 summarizes the names and uses of these subroutines. The subroutines are now described in detail with the required calling sequence given in each case.

CALL DEFREG(NREG,INTPC,INTX,INTY,INTPB) The argument list defines locations in the FORTRAN program that will receive the display parameter values when interrupts occur. All of the arguments are FORTRAN integers. This subroutine also initializes the assembly language interrupt processing support. The meanings of the argument list variables have been given above.

Table 1. Summary of FORTRAN interrupt processing subroutines

SURBROUTINE NAME	FUNCTIONAL DESCRIPTION
DEFREG	Initializes the system and defines graphic parameter's registers.
ENABLE	Used to enable or select those interrupts desired.
DISABL	Used to selectively disable any of the interrupts previously enabled.
INTON	Places all ENABLE'd interrupts into an active state.
RETURN	Re-sets interrupt conditions, and, if desired returns the program execution to the point of interruption.
WAIT	Generates a variable length programmable delay. It is used to wait for interrupts by setting the delay equal to infinity.

CALL ENABLE(ARG1,ARG2(,ARG3)) This subroutine is used to define the entry points (FORTRAN line numbers) for either of the interrupt types. ARG1 specifies the interrupts to be enabled while ARG2 (and ARG3 if required) defines the FORTRAN line number where the interrupt processing is to begin. If ARG1=1 then lightpen interrupts are enabled. If ARG1=2 then pushbutton interrupts are enabled. If ARG1=3 (ie:1+2) then both lightpen and pushbutton interrupts are enabled. ARG2 and ARG3 must be defined in a FORTRAN ASSIGN statement. This stores the address associated with a particular line number in the specified variable location. For example "ASSIGN 100 TO IADLP" causes the real memory address associated with line 100 to be stored in the variable IADLP. ARG3 is needed only when both interrupts are to be enabled (ie:ARG1=3). In this case it corresponds to the line number entry point for the pushbutton interrupts. ARG2, in this case, corresponds to the lightpen interrupt entry point. When only one interrupt is enabled ARG2 defines the line number entry point for the corresponding interrupt.

CALL DISABL(ARG1) The argument is composed indentically as the first argument of the ENABLE subroutine described above. The subroutine is used to selectively disable any previously ENABLE'd interrupts.

CALL INTON All interrupts that have been enabled are activated by the calling of this subroutine. It does not have an argument list.

CALL INTOF This subroutine de-activates all graphics' interrupts. Interrupts previously ENABLE'd remain so but are de-activated until the INTON subroutine is called.

CALL RETURN(ARG) This subroutine is used to terminate an interrupt processing sequence. The software block inhibiting entry into the FORTRAN interrupt handling sequence is removed. If ARG=Ø, the interrupted program is re-started from the point of interruption. If ARG=1 program control is passed to the FORTRAN statement following the subroutine call.

CALL WAIT(ARG) This subroutine is used to produce a variable length programmable delay. Return from the subroutine occurs after ARG milliseconds unless ARG=Ø in which case return never occurs. In the latter case the subroutine can be used to wait for interrupts.

This interrupt programming technique and the corresponding set of subroutines have been implemented and used successfully in the development of interactive graphics software. All of the graphics interrupt handling required in the development of the next technique discussed was programmed using this set of FORTRAN subroutines.

2. The Interactive Creation of Display Images: Normally graphics display layouts are specified by coding in a high level language, say FORTRAN, and the resultant code is compiled and run in order to test the "acceptability" of the corresponding display image. Typically several interations are required to produce the desired image, and consequently, this can be a time consuming phase in the development of graphics programs. The interactive method developed allows a user to sit in front of the display console and, via two modes of operation, create lines and text strings positioned as desired on the cathode ray tube screen. Immediate feedback as to the result of an operation allows him to detect and correct dynamically any mistakes. The modes of operation are;

i) Text mode: text command lines are entered and interpretively decoded to perform such operations as the drawing of lines, the creation and copying of subpictures, and the addition of text strings,

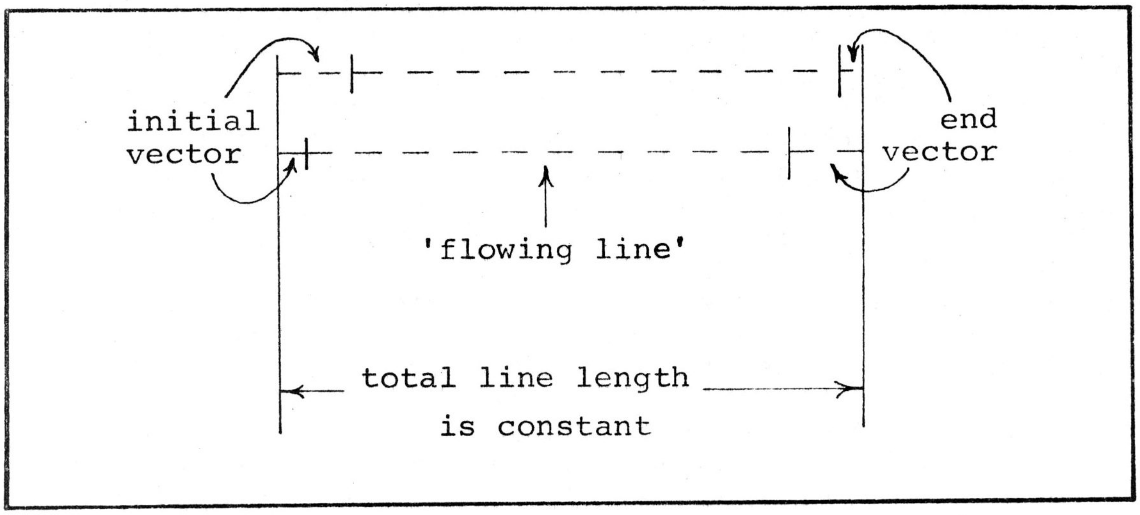
ii) Track mode: A tracking octagon and several push-button functions combine to provide for the interactive creation of sketchy images.

Text mode commands exist which provide for the creation of lightpen sensitive display parts (also referred to as "light buttons"). The image resulting from a display development session can be stored in a special disk or Dec-tape (a type of magnetic storage tape) file and can subsequently be reloaded and used by a graphics application program. This application program can use the FORTRAN interrupt processing technique to service the interrupts from the TEXT mode defined 'light buttons'. The net result of the integration of these two techniques is the PICADE system. The primary advantage achieved using PICADE is the quick, accurate, and easily mastered process for the generation of active display environments (ie: environments using display images with 'light buttons'). In some cases program size savings are achieved as the basic graphics drawing package will not be loaded into memory if there are no drawing functions performed at application's program run time.

Part B - Flowing Lines on the display

A package used to animate system or computer flow diagrams requires some method for indicating the flow of control from one box in the chart to the next. The technique used presents a dashed line connecting successive boxes and causes the dashed line to appear to be flowing in the same direction as the desired transfer of control indication. This illusion is created by causing the origin of the dashed line to move in the desired direction in real time. The origin is moved, in steps, a distance equal to a dash and a space of the given line and re-cycled through this sequence and thus the line appears to be flowing in the given direction. The rate of flow is a function of the step size and the time between steps. As the vectors drawn on the Graphics-15 are relative, the origin of the dashed line is moved by changing the length of a short initial vector. Another varying length vector is drawn after the end of the "flowing" line of a length such that the total line length is constant. (See the drawing).

Display file execution on the graphics-15 is synch'ed to the 60 c.p.s. line frequency. The graphics interrupt handler is entered every 1/60th of a second at the end of each complete execution of the display file. The initial and end vecotrs used to animate the flowing line are modified in length in the regularly entered graphics interrupt handler. The time between steps is therefore accurately determined. As an additional pair of these short varying length vectors is required for each additional line direction, eight pairs of lines are included. Thus flowing line illusions are possible in the eight basic vector directions (spanning a full 360^{o}). The initial and end vectors for a given line are accessed as subpictures. Thus all lines flowing in the same direction use the same initial and end vector pair.



This approach removes all support for the flowing line illusion to the interrupt handler and thereby facilitates the use of this feature at the graphics application program level. A "flowing line" drawing subroutine copies the initial vector as a subpicture, draws the desired dashed line, and then copies the end vector. In this manner a flowing line is created by one subroutine call specifying the line's length and direction (a number from 0-7). This feature will ultimately be integrated into a system which will animate system and computer flow diagrams.